VNIVERSIDAD NACIONAL
AVFNOMA DE
MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

AORTA: ARQUITECTURA DESACOPLADA Y ORIENTADA A ACCIONES PARA SOPORTAR COORDINACIÓN Y CONCIENCIA DE GRUPO EN SISTEMAS CSCW/CSCL

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN INGENIERÍA
(Computación)

P R E S E N T A:

PABLO MIGUEL OROZCO SUÁREZ

DIRECCIÓN:
DR. MANUEL ROMERO SALCEDO

CODIRECCIÓN:
DR. JUAN IGNACIO ASENSIO PÉREZ
DR. IOANNIS DIMITRIADIS DAMAULIS

MÉXICO, D.F. 2005.

a mis padres,
a mi hermana Carla y
a Susana.

Pablo Orozco.

Agradecimientos

A mis padres. Por el apoyo incondicional de toda una vida. Han sido ellos quienes me han inculcado el hambre de superación que me trajo hasta aquí y se los agradezco entrañablemente. Su ejemplo y motivación ha sido y será la llave de todos mis logros. Éste no es la excepción, les pertenece a ellos.

Al Dr. Juan Ignacio Asensio. Por sus valiosas ideas y por todo el esfuerzo que junto conmigo realizó. Sin duda tener en la dirección una persona con su capacidad y conocimientos ha sido un hecho decisivo para los logros alcanzados por este trabajo. Aprovecho para expresar la profunda admiración que siento por él.

Al Dr. Yannis Dimitriadis. Por haberme aceptado en su extraordinario equipo de trabajo, por su invaluable enseñanza y sobre todo por el gran apoyo y la confianza que me ha brindado siempre. Quiero decir que haber trabajado con una persona de su calidad científica y humana ha sido un verdadero privilegio para mí.

Al Dr. Manuel Romero. Por el apoyo y la confianza que ha depositado en mi durante todos estos años, pero sobre todo por su sincera amistad. En el curso de mi desarrollo profesional él ha sido una persona muy valiosa y se lo agradezco enormemente.

A la Universidad Nacional Autónoma de México. Por la inmejorable formación humana y profesional que me ha ofrecido durante todos estos años de estudio. Muy especialmente quiero agradecer a los profesores del Posgrado en Ciencia e Ingeniería de la Computación por su extraordinaria labor docente. Los méritos alcanzados por este trabajo en buena parte les corresponden.

Al grupo de investigación GSIC. Por sus consejos e ideas, por el reconocimiento que siempre hicieron de mi trabajo y sobre todo por su amistad. Con ellos (Alejandra, Bartolomé, Davinia, Eduardo, Guillermo, Iván, Luis Miguel y Miguel) he compartido una de las etapas más enriquecedoras de mi vida y les agradezco haberlo hecho posible.

Sólo me resta agradecer al CONACYT (Consejo Nacional de Ciencia y Tecnología de México), a la JCyL (Junta de Castilla y León de España), a la Universidad de Valladolid y al grupo de investigación GSIC por el financiamiento otorgado durante las diferentes etapas de mis estudios de posgrado.

De la enseñanza

No existe el hombre que pueda revelaros nada que no yazga aletargado en la aurora de vuestro conocimiento.

El maestro, que rodeado de sus discípulos camina por la sombra del templo, no os da de su sabiduría, sino más bien, de su fe y de su afecto.

Si en realidad es sabio, no os vedará el acceso a su saber, sino os conducirá mejor al umbral de vuestra propia inteligencia.

Khalil Gibran.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Motivación

El Aprendizaje Colaborativo Asistido por Computadora[1] (CSCL) es un paradigma de investigación que estudia el uso de las tecnologías de cómputo dentro del aprendizaje colaborativo [40]. Dicho de otra forma, para que un proceso de aprendizaje tenga lugar de forma colaborativa es necesario que exista un medio a través del cual los distintos participantes de este proceso puedan colaborar. Entre los diversos medios que pueden asistir a la colaboración en CSCL se hace énfasis en uno: artefactos tecnológicos interconectados.

Sin embargo, CSCL no es el único paradigma de investigación que estudia el uso de las tecnologías de cómputo para asistir la colaboración. Desde un punto de vista de tecnológico, CSCL tiene sus orígenes dentro de otro paradigma de investigación conocido como Trabajo Colaborativo Asistido por Computadora[2] (CSCW).

CSCW y CSCL son dos paradigmas íntimamente relacionados porque comparten muchos de sus principios y características. Sin embargo, no se debe perder de vista que entre ambos también existen diferencias claras: en CSCL el groupware[3] se diseña específicamente para soportar el aprendizaje; mientras que en el caso de CSCW el groupware se diseña pensando en soportar el trabajo. Por lo tanto, mientras CSCL pretende aumentar la carga cognitiva en los individuos para aumentar de esta forma su aprendizaje, CSCW intenta reducir esa misma carga cognitiva para aumentar la productividad del grupo.

[1] Del término en inglés *Computer Supported Collaborative Learning*.

[2] Del término en inglés *Computer Supported Collaborative Work*.

[3] Los sistemas groupware son sistemas de cómputo que asisten a grupos de personas interesados en una misma meta y que proveen una interfaz para interactuar en un ambiente compartido [20].

Desafortunadamente el desarrollo de sistemas CSCW/CSCL desde sus inicios se ha caracterizado por ser una tarea compleja. Esto se debe fundamentalmente a que los diversos actores que están inmersos en un proceso de colaboración pueden llegar a ser autónomos en términos de estrategias, heurísticas, perspectivas y conceptualizaciones. Manejar todo este dinamismo, inherente a cualquier escenario de colaboración, a través de un sistemas de software es tan costoso como complicado.

Debido a esto, el éxito de una aplicación CSCW/CSCL está íntimamente relacionado, entre otras cosas, con su capacidad para adaptarse o reutilizarse dentro de diversos escenarios de colaboración. Por ejemplo, si una aplicación CSCL que ha sido utilizada con éxito anteriormente sufre un cambio, en los objetivos de aprendizaje o el número de actores involucrados inicialmente, y no es capaz de adaptarse a este cambio se convierte en una aplicación inservible. En consecuencia, desde un punto de vista práctico, el esfuerzo y el costo que conlleva desarrollar esta aplicación no estará justificado.

Como una consecuencia obvia de la búsqueda de herramientas que permitan la creación de software más fácilmente adaptable y reutilizable, CSCW y CSCL, como muchas otras disciplinas, han enfocado su atención en un nuevo paradigma tecnológico para el desarrollo de software: Ingeniería de Software Basada en Componentes[4] (CBSE) [91].

CBSE constituye, al menos teóricamente, una solución potencial al problema de reutilización y adaptación del software. En primer lugar, es más fácil reutilizar partes (componentes) de un software que el software completo; se reutilizan sólo aquellas partes que tienen funcionalidades comunes a diversas aplicaciones. En segundo lugar, la adaptación de un software puede darse a través de una sustitución parcial; sólo aquellas partes (componentes) que no satisfacen los nuevos requerimientos son sustituidas.

Dentro del ámbito de los sistemas distribuidos algunos de estos conceptos y principios, íntimamente relacionados al uso de CBSE, han sido exitosamente aplicados [75]. Particularmente, el desarrollo de aplicaciones distribuidas a través del uso de marcos[5] ha encontrado en CBSE un complemento idóneo para ofrecer verdaderas soluciones de reutilización. Los marcos constituyen un diseño reutilizable (total o parcial) de un sistema. Este diseño se representa a través de un conjunto de elementos (componentes de software) y un esquema (patrones de diseño) que describe cómo esos elementos deben interactuar [12].

El dominio de los sistemas colaborativos no es ajeno al uso de marcos y componentes [3][26][29]. Por el contrario, la gama de servicios de colaboración que pueden reutilizarse a través del uso de un marco es variada y esto hace que los marcos constituyan un punto de inicio muy atractivo para el desarrollo de nuevas aplicaciones CSCW/CSCL.

[4]Del término en inglés *Component Based Software Engineering*.
[5]Del término en inglés *Frameworks*.

Desafortunadamente, no todo el soporte de colaboración que se requiere para el desarrollo de aplicaciones CSCW/CSCL ha sido tomado en cuenta por parte de los marcos. Mientras algunos servicios como el manejo de objetos compartidos, el manejo de sesiones o el manejo de grupos, son fuertemente soportados; otros servicios, no menos importantes como son la coordinación y la conciencia de grupo[6], cuentan con un soporte menor o incluso inexistente dentro de los marcos.

Precisamente, el caso de estudio de este trabajo aborda esta problemática como tema central *(proveer el soporte adecuado de coordinación y conciencia de grupo a marcos y aplicaciones CSCW/CSCL)* y presenta, como una posible solución: AORTA[7] (Arquitectura Desacoplada y Orientada a acciones para la Coordinación y la Conciencia de grupo).

AORTA es una arquitectura de software que ofrece, a través de un conjunto de componentes de software potencialmente integrables dentro de aplicaciones o marcos CSCW/CSCL existentes, soporte de: **coordinación a nivel de objetos y conciencia del espacio de trabajo.** El principal logro de AORTA es esconder, a los desarrolladores de aplicaciones CSCW/CSCL, la complejidad asociada al manejo de tareas de coordinación y conciencia de grupo.

A continuación se definen los conceptos *coordinación a nivel de objetos* y *conciencia del espacio de trabajo*:

Coordinación a nivel de objetos. Se se encarga de los accesos simultáneos o secuenciales de múltiples participantes al mismo conjunto de objetos [21]. Por ejemplo, en un editor colaborativo donde los participantes modifican un documento por turnos, la coordinación a nivel de objetos decide a qué participante corresponde el siguiente turno. Si un participante no tiene el turno e intenta modificar el documento la coordinación a nivel de objetos es responsable de prohibir la modificación del documento.

La coordinación a nivel de objetos se relaciona con la organización de las operaciones o acciones que son realizadas por usuarios sobre objetos compartidos; y no con la organización de procesos que realiza un sistema operativo. Tampoco debe confundirse la coordinación a nivel de objetos con la coordinación a nivel de actividades, pues esta última se encarga de manejar el flujo de tareas que los actores realizan cuando utilizan una herramienta colaborativa. La coordinación a nivel de objetos es un aspecto clave dentro de cualquier herramienta de colaboración ya que fortalece la sinergia que hace a los grupos de trabajo más productivos [21].

[6] Del término en inglés *Group Awareness.*

[7] Del término en inglés *Action-oriented decOupled aRchitecture for coordinaTion and Awareness.*

Conciencia del espacio de trabajo. Representa la información al momento sobre las interacciones de otros participantes con el espacio de trabajo compartido [32]. Si se retoma el ejemplo anterior del editor colaborativo, la conciencia del espacio de trabajo está dada por la información que deben tener todos los participantes sobre quién está modificando qué parte del documento en cada instante. La conciencia del espacio de trabajo compartido es un elemento de crucial importancia para las herramientas colaborativas ya que permite a los grupos manejar el proceso de trabajar de forma colaborativa [16]. Existen otros tipos de conciencia de grupo como la conciencia social o la conciencia de tarea entre otros; sin embargo, todos ellos salen del alcance de este trabajo. Por lo tanto, dentro del ámbito de este trabajo, el término conciencia de grupo siempre hace referencia a la conciencia del espacio de trabajo.

1.2. Antecedentes

1.2.1. GSIC: Grupo de Sistemas Inteligentes y Cooperativos

CSCL es un dominio de carácter multidisciplinar en el que confluyen dificultades inherentes a los dominios tecnológico (CS) y educativo (CL). Este hecho ha motivado la aparición de grupos de investigación interdisciplinares que coadyuvan sus conocimientos dentro del dominio CSCL y que pretenden lograr con esto aportaciones más significativas.

Un ejemplo de estos grupos es el GSIC (Grupo de Sistemas Inteligentes y Colaborativos). El GSIC es un grupo interdisciplinar formado dentro del ámbito universitario que está integrado por investigadores y estudiantes de las áreas de telemática, informática y educación de la Universidad de Valladolid.

El grupo GSIC colabora fuertemente con diversos grupos de investigación procedentes de diversas instituciones. Precisamente, el presente trabajo es producto un esfuerzo conjunto entre investigadores y estudiantes de la Universidad Nacional Autónoma de México (UNAM), el Instituto Mexicano del Petroleo (IMP) y el grupo de investigación interdisciplinar GSIC.

1.2.2. COSACO: Componentes de Software para Aplicaciones de Aprendizaje Colaborativo

El dominio educativo posee características muy peculiares que impactan directamente en el desarrollo del software para el aprendizaje. En general, los sistemas de software que asisten el aprendizaje adolecen de la flexibilidad que el dominio educativo les exige.

Los escenarios de aprendizaje se caracterizan por poseer un alto dinamismo que conlleva inexorablemente a la aparición constante de nuevos requerimientos. Por tal motivo, para los profesores y educadores es una necesidad primaria contar con sistemas de software flexibles que puedan ser adaptados de forma simple (incluso por ellos mismos) a diversos escenarios de aplicación.

A esta demanda que hace el dominio educativo la ingeniería de software ha ido respondiendo a través de la historia con diversas propuestas tecnológicas [75]. Sin embargo, todas éstas hasta ahora han quedado lejos de proveer soluciones verdaderas.

Ante la incapacidad de encontrar la solución a esta creciente necesidad, que no es sólo propia del software educativo sino de todo el software en general, ha surgido un nuevo paradigma tecnológico para el desarrollo de software: CBSE. Con la promesa de crear sistemas de software verdaderamente abiertos, flexibles y reutilizables CBSE ha cautivado la atención de la mayor parte de la industria y la investigación del software.

Tomando CBSE como el punto de partida, educadores y tecnólogos del grupo GSIC han realizado un proyecto de investigación, llamado COSACO[8] (Componentes de Software para Aplicaciones de Aprendizaje Colaborativo), con el objetivo general de determinar de que forma y en que grado CBSE puede contribuir a solucionar los problemas de falta de flexibilidad y reutilización de las aplicaciones CSCL. El presente trabajo de tesis constituye una línea de investigación más que se desprende del proyecto COSACO y pretende coadyuvar en el alcance de los objetivos generales de éste.

1.3. Objetivos

El presente trabajo de tesis tiene como objetivo principal:

El diseño e implementación de una arquitectura de software basada en componentes que ofrezca soporte de coordinación a nivel de objetos y soporte de conciencia del espacio de trabajo a marcos y aplicaciones CSCW/CSCL síncronas.

[8]El proyecto COSACO ha sido financiado por el Ministerio de Ciencia y Tecnología de España (TIC2000-1054).

OBJETIVOS ESPECÍFICOS DE COSACO
La generación, por parte de los pedagogos, de componentes educativos abstractos creados a partir del análisis de técnicas o situaciones CSCL
La provisión a los pedagogos del soporte y servicio adecuados para particularizar sus componentes a los requisitos educativos de un entorno concreto
La obtención de componentes de software concretos, a partir de los componentes abstractos anteriores y el diseño o adaptación de un entorno colaborativo que permita la creación de aplicaciones CSCL/CSCW basadas en tales componentes

Cuadro 1.1: Objetivos específicos del proyecto COSACO.

Dicho objetivo se enmarca dentro de los objetivos específicos del proyecto de investigación COSACO (ver cuadro 1.1) y para su alcance se plantea la siguiente metodología de trabajo:

1. Estudio del estado de arte de CSCL.

2. Estudio y análisis de los distintos enfoques de CBSE.

3. Diseño o adaptación de un entorno colaborativo (marco) que ofrezca soporte de coordinación a nivel de objetos y soporte de conciencia del espacio de trabajo a aplicaciones CSCW/CSCL basadas en componentes.

4. Selección de una aplicación CSCL, que servirá como base para la validación del entorno.

5. Desarrollo de un prototipo del entorno.

6. Validación de la funcionalidad del entorno sobre la aplicación CSCL.

7. Evaluación inicial del sistema dentro del marco del proyecto de investigación COSACO.

8. Elaboración de una publicación/artículo para un congreso internacional del área.

1.4. Estructura de la tesis

Este trabajo de tesis está estructurado de la siguiente forma:

Capítulo 1

El Capítulo 1 tiene el objetivo de introducir el trabajo de tesis. En éste se detalla cual ha sido la motivación que ha originado el trabajo, así como sus antecedentes y objetivos.

Capítulo 2

El Capítulo 2 realiza un estudio sobre el estado del arte de CSCL y su relación con algunos otros temas que son prioritarios para los fines de este trabajo. A continuación se listan dichos temas:

- Coordinación.
- Conciencia de grupo.
- Ingeniería de software Basada en Componentes (CBSE).
- Marcos para el desarrollo de aplicaciones.

Capítulo 3

Este capítulo, el Capítulo 3, detalla la creación de un marco conceptual que permita la integración de modelos de coordinación y conciencia de grupo en sistemas CSCW/CSCL.

Capítulo 4

Partiendo de las bases teóricas expuestas en el Capítulo 3, el Capítulo 4 propone el diseño detallado de una arquitectura de software, llamada AORTA, que ofrece servicios de coordinación y conciencia de grupo a aplicaciones y marcos CSCW/CSCL.

Capítulo 5

El Capítulo 5 describe las características más importantes del primer prototipo que se ha desarrollado de AORTA. Además, en este capítulo se detalla la utilización de dicho prototipo dentro de un primer caso de uso práctico como prueba de validación de su funcionalidad.

Capítulo 6

Las conclusiones y líneas de trabajo futuro de este trabajo de tesis se enmarcan dentro del Capítulo 6.

Apéndice A

Finalmente, en el Apéndice A, se presenta la Interfaz de Programación de AORTA.

Capítulo 2

Antecedentes

2.1. Aprendizaje Colaborativo Asistido por Computadora

El campo del Aprendizaje Colaborativo Asistido por Computadora estudia el uso de las tecnologías de computo dentro del aprendizaje colaborativo.

Tim Koschmann [40].

2.1.1. El proceso de aprendizaje

El aprendizaje puede ser entendido como la acción y el efecto de generar conocimiento en los individuos. Sin embargo, tratar de interpretar de que forma se genera este conocimiento y cuáles causas lo propician es una tarea complicada y largamente estudiada. Prueba de ello es la existencia de diversas teorías, incluso contradictorias, que ofrecen su interpretación particular sobre el proceso de aprendizaje.

Existen dos grandes grupos en que clasifican las teorías sobre el proceso de aprendizaje:

- Teorías conductistas.
- Teorías cognitivistas.
- Teorías híbridas.

Las teorías conductistas sostienen que dentro del proceso de aprendizaje el individuo es sólo un ente perceptivo de información que recibe conocimiento a través de su interacción con el medio externo. Por lo tanto los individuos no son capaces de generar su propio conocimiento. El condustismo está fundado en buena parte en los estudios de *Iván Petróvich Pavlov*, quien es considerado uno de sus precursores más destacados. *Pavlov* afirma que los individuos son sólo reacción y la base de su estudio debe ser su conducta; en consecuencia el aprendizaje se reduce simplemente a la modificación de conductas.

A raíz de estas aseveraciones, interpretaciones más modernas sobre cómo se da el proceso de aprendizaje han cuestionado fuertemente a las teorías conductistas. La aparente incapacidad del conductismo para hacer frente a las expectativas educativas más modernas es lo que ha dado lugar a las teorías cognitivistas.

Las teorías cognitivistas, por otro lado, sostienen que los individuos sí son capaces de generar este conocimiento y afirman que la construcción del conocimiento está determinada, en buena parte, por procesos cognitivos del propio individuo.

A continuación se estudian con mayor detalle las principales características de ambos grupos de teorías así como su relación con el aprendizaje colaborativo y con CSCL.

2.1.2. Teorías del aprendizaje

Conductismo

John B. Watson, fundador del conductismo, postuló a principios del siglo XX que la psicología, ciencia que estudia los procesos mentales en personas y en animales, debía ocuparse únicamente del comportamiento externo del individuo, evitando de esta forma la ingerencia en el estudio de conceptos que no pueden ser estimados objetivamente como es el caso del propio conocimiento.

Partiendo de la base conductista postulada por *John B. Watson*, *B.F. Skinner*, considerado a la fecha el máximo representante del conductismo, afirmó que los procesos mentales no determinaban la conducta de los individuos, y, por lo tanto, el aprendizaje podría ser explicado como la consecuencia única de los estímulos externos que recibe cada individuo [17].

El principal postulado del paradigma conductista es:

> Todos los procesos mentales, entre ellos el aprendizaje, no determinan la conducta de los individuos. La conducta es el producto del propio condicionamiento de cada individuo. Los seres vivos son entes biológicos y no actúan conscientemente; más bien reaccionan a estímulos [10].

Las principales características de las teorías conductistas son:

- El individuo no es capaz de crear conocimiento, este proceso es sólo consecuencia de su relación con el medio; siendo éste, el propio medio, la única causa de que el proceso de aprendizaje tenga lugar.
- Las particularidades de cada individuo para asimilar el proceso de aprendizaje no existen. Dicho proceso se da de igual forma para cualquier ser humano. Incluso se sostiene que el proceso de aprendizaje es el mismo en hombres, animales y máquinas.
- Dentro del proceso de aprendizaje no intervienen procesos cognitivos.

Cognitivismo

> El cognitivismo es un término que sirve para amparar a varias ciencias distintas que intentan comprender el funcionamiento de la mente [66].

El paradigma cognitivista, como se comentó anteriormente, surge fundamentalmente como una oposición al conductismo. Las teorías cognitivistas sostienen, al contrario de las conductistas, que los procesos cognitivos de los individuo influyen los procesos mentales de éstos (un ejemplo de estos procesos es la generación de conocimiento y por consiguiente otros ejemplos podrían ser la propia conducta o el comportamiento). Otra característica importante del paradigma cognitivista es que considera a los individuos entes capaces de generar su propio conocimiento.

El cognitivismo a dado lugar a numerosas teorías. Algunas de las más importantes, a partir de su relación con el campo CSCL, ámbito de estudio del presente trabajo, son (ver cuadro 2.1):

- Teoría constructivista.
- Psicología genético cognitiva.
- Teoría del aprendizaje significativo.
- Teoría del aprendizaje social.
- Teoría del aprendizaje situado.

2.1.3. Teorías que subyacen el Aprendizaje Colaborativo

Teoría constructivista

La teoría constructivista asegura que el aprendizaje es un proceso activo en el que el individuo no es un simple receptor de información, cómo afirman las teorías conductistas, sino que utiliza sus conocimientos y experiencias previas para construir nuevo conocimiento.

Para la teoría constructivista el conocimiento es siempre una interacción entre la nueva información que se presenta al individuo y lo que éste ya sabía. Partiendo de este principio, la teoría constructivista afirma que aprender es construir modelos capaces de interpretar la información que se recibe [68].

Por lo tanto, el individuo, en el momento de enfrentarse al proceso de aprender, antepone sus esquemas previos, es decir, su representación interna de una situación en particular, condicionando así el aprendizaje. La interacción con la realidad hace que estos esquemas vayan cambiando, y, de esta forma, cuanto mayor sea la interacción mayor será la posibilidad de mantener esquemas amplios y abiertos [66].

Psicología genético cognitiva

La teoría de *Jean Piaget*, también conocida como psicología genético cognitiva, sostiene que el aprendizaje está condicionado por las estructuras iniciales de cada individuo. Siendo además este mismo aprendizaje la causa posterior de su modificación. A través de la modificación de sus estructuras iniciales los individuo son capaces de alcanzar aprendizajes más complejos cada vez. La teoría de *Piaget* fortalece fundamentalmente el aprendizaje amplio y significativo de los conceptos. Considerando que es tan importante entender las partes aisladas como el todo significativo.

Teoría del aprendizaje significativo

La teoría del aprendizaje significativo, mejor conocida como el Aprendizaje significativo de *Ausubel*, plantea que el conocimiento nuevo debe partir de una relación con el conocimiento existente. Dicho de otra forma, lo que la teoría de *Ausubel* enfatiza es la posibilidad de adquirir conocimiento a través de relaciones cognitivas del propio individuo. De tal forma que el individuo genere conciencia sobre el nuevo conocimiento adquirido. *Ausubel* considera que un *puente cognitivo* es la relación o el medio que permite a un individuo pasar de una primera fase de conocimiento a una segunda fase de conocimiento (mayor a la primera). Los puentes cognitivos, afirma Ausubel, contribuyen enormemente al proceso de aprendizaje.

TEORÍAS
Teoría constructivista
Psicología genético cognitiva
Teoría del aprendizaje significativo
Teoría del aprendizaje social
Teoría del aprendizaje situado

Cuadro 2.1: Teorías que subyacen al Aprendizaje Colaborativo

Teoría del aprendizaje social

Lev Semenovich Vigotsky sostiene que los seres humanos también pueden aprender por medio de interacciones sociales. Dicho de otra forma, cuando un individuo percibe de forma directa o indirecta lo que le sucede a otros individuos o la información que estos otros individuos reciben, éste es capaz de generar conocimiento propio. Esta aseveración constituye el principal postulado de la teoría del aprendizaje social.

La teoría del aprendizaje social enfatiza la importancia de la observación e imitación de los comportamientos ajenos. Además, se destaca el concepto de *zona de desarrollo próximo* como el proceso de aprendizaje que parte de una base cercana al propio individuo con el objetivo de facilitar a éste la adquisición del nuevo conocimiento. En una analogía con la distancia, un conocimiento totalmente nuevo representa punto lejano para el individuo. La zona de desarrollo próximo está representada por los puntos cercanos en distancia; a los que el individuo puede acceder fácilmente.

Teoría del aprendizaje situado

La teoría del aprendizaje situado postula que el proceso de aprendizaje está fuertemente influenciado por el propio contexto en el que éste se da. Esta teoría considera que el contexto debe tener matices realistas para permitir al individuo desarrollar sus habilidades y capacidades de aprender de forma óptima.

El conjunto de teorías detallado a lo largo de esta sección constituye la base teórica sobre la que fundamenta el aprendizaje colaborativo. De éste y de su relación con CSCL se habla a continuación.

2.1.4. Aprendizaje Colaborativo

El término colaboración se refiere a cualquier actividad que dos o más individuos realizan juntos [93].

En algunas disciplinas relacionadas con el aprendizaje, el término colaboración enfatiza ideas de corresponsabilidad y compromiso compartido para la construcción del conocimiento. Por tal motivo, la definición anterior podría no ser suficiente para reflejar lo que el término colaboración significa cuando tiene por objetivo establecer un proceso de aprendizaje en los individuos.

Desde esta segunda lectura una definición del término colaboración podría ser alguna de las siguientes:

> Un proceso de enculturación que ayuda a los estudiantes a hacerse miembros de comunidades de conocimiento cuya propiedad común es diferente a la propiedad común de las comunidades de conocimiento a las que pertenecían antes [79].

> Una actividad coordinada y sincronizada, resultado de un intento por construir y mantener la concepción compartida de un problema [93].

Adoptando esta segunda definición es posible definir el término *aprendizaje colaborativo* de la siguiente forma:

> Conjunto de estrategias de enseñanza que dependen de la interacción de un pequeño grupo de aprendices, siendo ésta la característica central de las tareas de aprendizaje en el aula [15].

> Forma de organización de la clase en la que los estudiantes trabajan en pequeños grupos, ayudándose unos a otros en el aprendizaje de los contenidos [81].

> Proceso social que ayuda a los estudiantes a llegar a ser miembros de una comunidad de conocimiento [7].

> Compromiso mutuo de participación en un esfuerzo coordinado para resolver un problema de forma conjunta [74].

Como evidencian todas estas definiciones proveer al término aprendizaje colaborativo de una definición universal es y ha sido hasta ahora una tarea imposible. Sin embargo, más allá de estas definiciones el aprendizaje colaborativo es una disciplina de estudio, perteneciente al dominio de la educación, que se deriva de la necesidad de entender el proceso de colaborar cuando éste tiene como objetivo generar un proceso de aprendizaje.

Después de casi un siglo de estudio, diversas líneas de investigación sostienen que las formas de aprendizaje colaborativo son más efectivas que las individuales o las competitivas [36]. Sin embargo, también existe otro tanto que contradicen este hecho.

2.1.5. Aprendizaje Colaborativo Asistido por Computadora

¿Qué es el Aprendizaje Colaborativo Asistido por Computadora?

Para que el aprendizaje colaborativo tenga lugar es necesario que exista un medio a través del cual la colaboración pueda darse entre los distintos participantes del proceso. Entre los diversos medios que podrían asistir a la colaboración en este trabajo se hace énfasis en uno: artefactos tecnológicos interconectados (por ejemplo computadoras). El aprendizaje colaborativo asistido por un conjunto de artefactos tecnológicos interconectados da lugar a una disciplina de estudio conocida como CSCL (Aprendizaje Colaborativo Asistido por Computadora). CSCL está definido como un conjunto de artefactos tecnológicos interconectados, y, el conjunto de técnicas, que a través de éstos se utiliza para asistir el aprendizaje colaborativo.

Un hecho interesante relacionado con el Aprendizaje Colaborativo Asistido por Computadora es el propio significado de sus siglas. En realidad, a pesar de que la traducción del término al idioma español es aceptada por la propia comunidad hispana de CSCL, la palabra *colaborativo* no forma parte del idioma[1].

Para el caso del idioma inglés, considerando que la palabra *Collaborative* es correcta, tampoco es claro el significado de ésta sobre los términos *Cooperative o Collective*. Prueba de ello es que durante el primer congreso CSCL el término estaba propuesto como *Computer Supporte Cooperative Learning* [41]. De igual forma, posterior a este hecho, se propone dar a CSCL el significado de *Computer Supported Collective Learning* [11].

[1] Palabras con un significado cercano dentro del idioma español podrían ser: colectivo o cooperativo.

Teorías educativas que sustentan el CSCL

Los sistemas CSCL están basados en la premisa que las computadoras pueden proporcionar sistemas que faciliten procesos y dinámicas grupales para lograr el aprendizaje.

Al mismo tiempo, esta premisa sustenta su hipótesis en una serie de teorías educativas (todas ellas estudiadas anteriormente):

1. Teoría constructivista.
2. Teoría del aprendizaje social.
3. Teoría del aprendizaje significativo.
4. Teoría del aprendizaje situado.

En CSCL se afirma, como principal base teórica, que la colaboración y el trabajo en grupo hacen posible la construcción del conocimiento (**Teoría del aprendizaje social**). Hecho que va más allá de un ámbito de conductas y que hace referencia a los individuos como entes de aprendizaje complejos, influenciados por particularidades tanto propias como provenientes de su entorno (**Teoría constructivista**). Además, se afirma que para facilitar la construcción de conocimiento en los individuos es necesario fijar un objetivo que sea significativo para el propio individuo (**Teoría del aprendizaje significativo**). Siendo necesario también un contexto adecuado, como puede ser una tarea de la vida real o un problema que esté relacionado con el mismo individuo que aprende (**Teoría del aprendizaje situado**).

Las principales afirmaciones teóricas que sustenta el paradigma CSCL se pueden resumir de la siguiente forma [39]:

- El conocimiento se construye, no se transmite. La construcción del conocimiento es un proceso natural. Cuando los seres humanos se enfrentan a algo desconocido, su inclinación natural les lleva a utilizar todo aquello que ya conocen para tratar de determinar qué significa. El trabajo de los educadores no es impartir conocimiento, sino proporcionar a los alumnos experiencias para que éstos construyan el suyo propio, y guiarlos, de esta forma, en el proceso de creación de conocimiento.
- La construcción del conocimiento surge como resultado de una actividad. No se puede separar el conocimiento de las cosas ni de las experiencias que se tienen con ellas.
- El conocimiento adquirido está influido por el contexto en el que se da el proceso de aprendizaje. El conocimiento que se genera como resultado de una actividad incluye información del contexto en que tuvo lugar

la experiencia de aprendizaje. Por lo tanto, cuanto más directamente se experimenten las cosas, más conocimiento seremos capaces de extraer de ellas.

- El conocimiento se encuentra en la mente del alumno. El proceso de construcción del conocimiento está condicionado por la percepción del mundo físico que posea el alumno. Dado que cada individuo tiene un conjunto diferente de experiencias, cada persona tendrá también una combinación propia y original de opiniones acerca del mundo. Podemos hablar a otros individuos de nuestras experiencias, pero el conocimiento que estos construyan a partir de ellas será la consecuencia de una interpretación personal de nuestras experiencias basada en las suyas propias.
- Existen múltiples perspectivas del mundo. Dado el hecho que es imposible que dos personas posean el mismo conjunto de experiencias, es así mismo imposible que construyan un conocimiento idéntico a partir de una misma experiencia.
- La construcción del conocimiento surge como consecuencia de una discrepancia entre lo que ya se conoce y lo que se observa en un momento determinado.
- La construcción del conocimiento requiere la articulación, expresión o representación de lo que se ha aprendido. Aunque la actividad es una condición necesaria para la construcción del conocimiento, no es suficiente. Para la construcción de un conocimiento útil, los alumnos necesitan pensar sobre lo que han hecho y expresar lo que esto significa.
- El conocimiento se puede compartir con otros individuos. Esto significa que el aprendizaje puede surgir, por ejemplo, como consecuencia de una conversación.
- El proceso de aprendizaje es un proceso social.
- Nuestro conocimiento está influenciado por el conocimiento y las creencias de las personas de nuestro entorno.

2.2. Groupware

2.2.1. Groupware y CSCW (Trabajo Colaborativo Asistido por Computadora)

El término *groupware* deriva de la conjunción de dos palabras: *GROUP* y soft*WARE*. Por tal motivo este término es comúnmente referido para denotar cualquier tipo de aplicación que soporta el trabajo de un grupo de usuarios. Sin embargo, esta primera clasificación no es universalmente aceptada.

Otra definición, y una de las más citadas en la literatura, propone clasificar mediante el término groupware sólo a aquellos sistemas de computo (hardware o software) que asisten a grupos de personas que tienen una tarea o meta común:

> Los sistemas groupware son sistemas de computo que asisten a grupos de personas interesados en una misma meta y que proveen una interfaz para interactuar en un ambiente compartido [20].

Adoptando está definición, y según mi punto de vista, el término groupware consiste en cualquier sistema de computo orientado, específicamente, a asistir grupos de gente (no individuos) que comparten un mismo objetivo. Esta definición excluye cualquier otra aplicación multiusuario donde los usuarios no compartan un mismo objetivo.

El término CSCW (Trabajo Colaborativo Asistido por Computadora), por otra parte, hace referencia a un campo multidisciplinar de investigación científica. CSCW es una disciplina científica que motiva y valida el diseño de groupware. Dicho de otra forma, es una ciencia que se encarga de describir cómo debe realizarse el desarrollo de aplicaciones groupware.

A CSCW le atañe el estudio y la teoría de cómo la gente trabaja de forma conjunta y cómo las aplicaciones groupware afectan este comportamiento (efectos psicológicos, sociales y organizacionales). CSCW, por lo tanto, es tecnológicamente independiente pero socialmente dependiente. Parte de la observación de cómo la gente interactúa para colaborar y a través de esto determina los lineamientos en los que se debe desarrollar la tecnología.

CSCL al igual que CSCW es un campo multidisciplinar de investigación científica. CSCL está íntimamente relacionado con CSCW porque ambos campos comparten muchos de sus principios y características. Sin embargo existe una clara distinción entre ambos campos que no debe perderse de vista. En el caso de CSCL el groupware se diseña específicamente para asistir el aprendizaje mientras que en el caso de CSCW el groupware se diseña pensando en soportar el trabajo.

Ambas disciplinas deben soportar la colaboración de un grupo, y en este sentido ambas se benefician al compartir su entendimiento de cómo la tecnología puede soportar mejor las interacciones de grupos. Sin embargo, aunque el aprendizaje y el trabajo son una consecuencia común en ambas disciplinas el objetivo de realizar esta colaboración es distinto para cada una. CSCW soporta la colaboración con el objetivo de realizar una tarea de forma conjunta. Ciertamente esto permite que exista un proceso de aprendizaje en los participantes pero es incidental y no es el resultado de un causal pedagógico. Para CSCL el objetivo de que un grupo colabore es netamente generar un proceso de aprendizaje.

Otra diferencia clara entre ambas es que CSCL pretende aumentar la carga cognitiva en los individuos para aumentar de esta forma su aprendizaje. CSCW, en cambio, intenta reducir esa misma carga cognitiva para aumentar de esta forma la productividad del grupo.

2.2.2. Taxonomías de los sistemas groupware

Taxonomías generales.

Una primera taxonomía propuesta para los sistemas groupware propone hacer su distinción de acuerdo al espectro de tiempo en que los usuarios interactúan. Esta taxonomía clasifica al groupware en dos grandes grupos:

- Groupware síncrono. Sistemas groupware que soportan actividad de usuarios de forma simultánea o concurrente.
- Groupware asíncrono. Sistemas groupware donde la actividad de usuarios distintos se da en periodos de tiempo también distintos.

En una segunda taxonomía se propone distinguir el lugar físico dónde tiene lugar la interacción de los usuarios. Para este caso, se tienen otras dos clasificaciones para los sistemas groupware:

- Cara a cara: Son aquellas herramientas que soportan la interacción grupal en el mismo lugar físico.
- Remotos: Son aquellas herramientas que soportan la interacción grupal a distancia.

Una de las taxonomías de groupware más aceptadas [20] se deriva, precisamente, de la combinación de variables tiempo y espacio. En esta tercera se distinguen cuatro tipos de sistemas groupware:

- Mismo tiempo - Mismo lugar: Interacción cara a cara.
- Mismo tiempo - Diferente lugar: Interacción distribuida síncrona.
- Diferente tiempo - Mismo lugar: Interacción asíncrona.
- Diferente tiempo - Diferente lugar: Interacción distribuida asíncrona.

LUGAR - TIEMPO	
Mismo - Mismo	Mismo - Diferente
Diferente - Mismo	Diferente - Diferente

Cuadro 2.2: Taxonomía Lugar - Tiempo

EJEMPLOS	
Aula de clases	Laboratorios y bibliotecas
Enseñanza remota	Educación a distancia

Cuadro 2.3: Ejemplos para la taxonomía Lugar - Tiempo

Taxonomías propias de los sistemas CSCL

Existen algunas taxonomías que son particulares de los sistemas CSCL. Éstas utilizan bases pedagógicas para sus clasificaciones y no son aplicables a ningún otro tipo de sistema groupware. A continuación se listan dos ejemplos.

La primera taxonomía, que se muestra a continuación, se basa en una aproximación docente y propone cinco distinciones para aplicaciones CSCL [11]:

1. El tutorial. Consiste en la presentación de multimedios a los aprendices. Posterior a la exposición el enseñante realiza y recibe preguntas al respecto.
2. La resolución de problemas. Los aprendices trabajan de forma conjunta para alcanzar la solución a alguna cuestión planteada. El enseñante funge exclusivamente como observador. Algunas herramientas que asisten a los alumnos en la solución de la cuestión: procesadores de texto herramientas de dibujo, programas para cálculos matemáticos, etc. Todos ellos se distinguen por ser aplicaciones monousuario.
3. La simulación o micromundo. Como su nombre sugiere, en este caso los estudiantes interaccionan con objetos dentro de un mundo simulado. El objetivo principal de esta técnica es lograr el conocimiento a través de la propia experiencia de los aprendices.
4. El debate. Esta basado en aplicaciones que permiten la comunicación, como foros, mensajería instantánea, conferencia de voz; además de aplicaciones que permiten la organización del debate, como herramientas de mapas conceptuales, esquemas, políticas de turnos.
5. El modelado. Es muy similar a un micromundo salvo porque en este tipo de aplicaciones es posible la propia modificación del micromundo mediante la edición de parámetros.

Una segunda clasificación, producto de la participación de pedagogos y tecnólogos del grupo GSIC, propone un conjunto formado por tres técnicas de aprendizaje. Es importante mencionar que esta clasificación no es exhaustiva; y las tres técnicas que se ilustran a continuación representan sólo un conjunto menor del total de técnicas de aprendizaje existentes.

1. Rompecabezas o *Jigsaw*. Esta técnica consiste en formar grupos y dividir la información para la solución del problema de manera que cada grupo tenga que resolver el mismo problema y cada integrante tenga una porción distinta de la información para resolver el problema. Entre todos los integrantes de cada grupo se construye la solución completa del problema.

2. Simulación. En esta técnica cada alumno asumirá el papel de un rol, usualmente definido por el profesor. El alumno deberá actuar en consecuencia a su propio rol y a las características de éste dentro de un escenario o situación que deberá también ser prefijado. Esta técnica es eficaz para trabajar habilidades de negociación y persuasión.

3. Pirámide o bola de nieve. En este caso el profesor plantea un problema y cada alumno deberá elaborar una lista de ideas para resolverlo. Posteriormente el alumno se juntará con un segundo compañero y, a partir de las dos listas, elaborarán una única lista con las mejores ideas. Más tarde, cada pareja se junta con otra pareja y harán lo mismo. Así sucesivamente hasta formar dos grandes grupos que deberán poner en común las mejores ideas. Esta técnica es útil para la generación de ideas.

2.2.3. Características de los sistemas groupware síncronos

El desarrollo de sistemas groupware es una actividad compleja. Estos sistemas, fundamentalmente los que permiten interacción síncrona, proporcionan espacios de trabajo compartidos potencialmente caóticos. Tener múltiples usuarios con la libertad de generar interacciones de forma libre en cualquier instante de tiempo da lugar a ambientes altamente dinámicos. Manejar este dinamismo de forma natural, para no sesgar abruptamente la colaboración, pero también de forma armónica, para lograr que las interacciones confluyan consecuentemente en un mismo punto, es la clave de una colaboración efectiva.

Para lograr esta efectividad en materia de colaboración los sistemas groupware deben atender una serie de requerimientos básicos que los distinguen de cualquier otro tipo de sistema de software. Estos requerimientos se derivan de necesidades propias de la misma colaboración. Solventar cada uno de estos requisitos permite que los sistemas CSCW/CSCL ofrezcan a sus usuarios los servicios necesarios para poder llevar acabo una colaboración efectiva.

Algunos de estos servicios son:

- Coordinación.
- Conciencia de grupo.
- Objetos compartidos.
- Sesiones.
- Roles.

De la lista anterior, se destacan los dos primeros por su relación con el tema de investigación de este trabajo. Por ese mismo motivo en las siguientes secciones se profundiza más sobre ambos.

La coordinación de actividades y la conciencia de lo que hacen los otros participantes son elementos inherentes a cualquier escenario de colaboración físico o virtual. Sin embargo, la dificultad de desarrollar y mantener servicios de coordinación y conciencia de grupo en sistemas groupware es particularmente compleja [33].

2.3. Conciencia de grupo

2.3.1. ¿Qué es la conciencia de grupo?

Los sistemas groupware síncronos permiten a diversos actores desde distintos lugares colaborar en la realización de un trabajo durante el mismo intervalo de tiempo [1].

A partir de esta afirmación se puede identificar el trabajo como la meta común de un grupo de actores que interactúan a través de la ejecución individual de acciones. Por lo tanto, desde una perspectiva global cada una de éstas acciones deberá coadyuvar en el logro de la misma meta.

Sin embargo, encausar acciones independientes hacia una meta común representa una tarea no trivial debido a que la naturaleza de cualquier escenario de colaboración es caótica y resulta casi imposible predecir de qué forma y cuándo tendrán lugar las acciones o los cambios de estado de éste.

En los escenarios de colaboración físicos las acciones se produce de forma natural (cara a cara). Hecho clave ya que permite en todo momento proveer al grupo de actores que participan de una información basta sobre las diferentes acciones que acontecen. Este conocimiento sobre las actividades que acontecen es vital para la efectividad de la colaboración [16].

Proveer de este conocimiento, al grupo de actores que participan en un el mismo espacio de trabajo (físico o virtual) les permite generar una *conciencia de grupo*. Es a través de esta conciencia como cada actor puede situar con efectividad sus acciones dentro de un contexto de acciones globales.

> La conciencia de grupo es el entendimiento de las actividades de los otros; ya que son estas actividades externas las proveen un contexto para las actividades propias [16].

La conciencia de grupo juega un papel fundamental dentro de la colaboración [16]. Permite la coordinación de tareas y recursos, además, constituye el puente de enlace entre actividades individuales y compartidas. Cuando un individuo tiene conciencia de lo que acontece en su espacio de trabajo puede utilizar este conocimiento para anticiparse a las acciones de otros, ayudarlos con sus tareas, etc. [31].

2.3.2. Tipos de conciencia de grupo

En [32] se propone clasificar la conciencia de grupo de acuerdo a los diferentes tipos de interacción que pueden sucitarse[2] en un espacio de trabajo compartido:

- Conciencia social (*Social awareness*).
- Conciencia de tarea (*Task awareness*).
- Conciencia de concepto (*Concept awareness*).
- Conciencia del espacio de trabajo (*Workspace awareness*).

Conciencia social

Conciencia sobre el contexto social. Para saber cómo interactuar con un grupo un individuo debe conocer: el rol que tiene dentro del contexto social y los roles que tienen el resto de los individuos. A partir de haber generado esta conciencia el individuo puede actuar consecuentemente a ello.

Una característica del *Social Awareness* es que se genera de forma interna al individuo por lo que el soporte que se puede ofrecer deberá estar implícito en cada aplicación groupware.

[2] Estas interacciones son situadas en un contexto de aprendizaje colaborativo.

Conciencia de tarea

Conciencia o conocimiento sobre cómo deben completarse las tareas que tienen lugar en el escenario de colaboración. El individuo se cuestiona qué sabe y qué sabe el resto del grupo acerca de la tarea. A partir de esto identifica las herramientas que se requieren, el tiempo, y con base en eso estructura la realización de la tarea.

Conciencia de concepto

Es un conocimiento conceptual sobre lo que se hace y lo que se sabe. Esto permite al individuo enlazar conceptos para la generación de nuevas ideas. Por ejemplo el individuo puede generar hipótesis o predicciones sobre tareas o hechos futuros que desconoce a partir del conocimiento que posee.

Concept awareness y *Task Awareness* son temas fuertemente estudiados dentro del dominio CSCL. Por ejemplo, aplicaciones que guían de forma didáctica de que forma deben realizarse tareas específicas que pretende generar aprendizaje en el individuo.

Conciencia del espacio de trabajo

Es el conocimiento sobre las interacciones que acontecen y las que ya han acontecido en el espacio de trabajo. ¿Dónde trabajan los otros individuos?, ¿qué hacen? o ¿qué han hecho?

La conciencia del espacio de trabajo es un concepto clave dentro de los sistemas groupware síncronos [30]. La efectividad de la colaboración depende en gran parte de ésta. Estudiar de qué forma generar, capturar y explotar la información relacionada con la conciencia del espacio de trabajo se ha convertido un problema clásico de los dominios CSCW/CSCL. La siguiente sección presenta un estudio más minucioso sobre este tema.

2.3.3. Conciencia del espacio de trabajo

> La conciencia del espacio de trabajo es el entendimiento de las interacciones de otra persona con el espacio compartido. A un nivel simple esto involucra el conocimiento de quienes están presentes, dónde están ellos trabajando y lo que están haciendo [31].

> El conocimiento global de una persona sobre las interacciones de otros que acontecen dentro de un espacio de trabajo compartido es lo que se conoce como *workspace awareness* o conciencia del espacio de trabajo [30].

La generación de un conocimiento global de las interacciones juega un papel fundamental dentro de la colaboración ya que permite hacer contribuciones de forma individual que sean relevantes dentro de una actividad grupal (a través del conocimiento de las interacciones que acontecen es posible evaluar las acciones de cada individuo con respecto a logros y progresos de todo el grupo).

En este mismo sentido para un sistema groupware resulta fundamental la cantidad de información que es capaz de proveer a los usuarios sobre el total de interacciones que acontecen. A pesar de esto uno de los problemas más típicos con los sistemas groupware es que esta información no suele ser suficiente. En cambio, cuando la gente interactúa cara a cara se tienen una gran variedad de elementos que ayudan a generar y percibir toda la esta información.

Esta información compartida sobre las interacciones que acontecen permite generar una conciencia de lo que sucede en el espacio de trabajo: *workspace awareness*.

La conciencia del espacio de trabajo implica diversos tipos de conocimiento sobre el resto de los individuos que están dentro del mismo escenario de colaboración y sobre las actividades que éstos llevan acabo. Este conocimiento además está íntimamente relacionado con un conjunto de actividades vitales para lograr un enlace efectivo entre trabajo individual y trabajo de equipo: coordinación de actividades, simplificación de la comunicación verbal, proveer asistencia, anticipación a las acciones de otros [31].

Existen diversos estudios cualitativos que demuestran el impacto que tiene la integración, dentro de los sistemas groupware síncronos, de mecanismos eficientes para proveer información relacionada con la conciencia del espacio de trabajo [16][23][31]. Sin embargo, llevar a cabo esta tarea es complicado debido a que toda la información se genera de forma dinámica y en distintos puntos del escenario de colaboración [33]. En consecuencia, para los sistemas groupware, proveer servicios para fortalecer la conciencia del espacio de trabajo es comúnmente una tarea costosa.

Aunque algunos sistemas groupware sí proveen mecanismos específicos [73] para fortalecer la conciencia del espacio de trabajo sus soluciones, hechas a la medida, tienen muy pocas posibilidades de reutilizarse y esto origina un nuevo problema dado el alto costo de diseño, implementación y mantenimiento que tienen.

Existen también otras herramientas [86][76][26][65] que sí permiten simplificar y reutilizar partes o servicios de los sistemas groupware. Sin embargo, como se verá más adelante, éste comúnmente no es el caso de la conciencia del espacio de trabajo.

2.4. Coordinación

2.4.1. ¿Qué es la coordinación?

Entender la *coordinación* a partir de un enfoque teórico sugiere, como primer paso, recurrir a la propia teoría de la coordinación. La teoría de la coordinación, entonces, está representada por un conjunto de principios, resultado de años de investigación en diversas disciplinas, que describen cómo actividades diferentes pueden ser coordinadas. Esta primera explicación sugiere entender el término coordinación como una actividad. Pero a partir de esta primera afirmación que dice que la coordinación es una actividad surge la primera cuestión:

¿Cómo se diferencia la coordinación del resto de las actividades?

A diferencia de la mayor parte de las actividades, [34] afirma, la coordinación puede distinguirse del resto por:

- La coordinación es un requerimiento, no en sí misma, pero sí a causa de otras actividades que la requieren.
- Todos los participantes, sin excepción, deben formar parte en el esfuerzo por alcanzar la coordinación.
- La coordinación no genera un producto, sino que sirve para establecer relaciones entre las actividades y los productos generados a partir de éstas.
- La coordinación no tiene un propósito independiente sino que es un requisito para el cumplimiento de otros propósitos.

A partir de estos principios podría proponerse una primera definición, de carácter intuitivo, sobre el término *coordinación*:

> Acto del comportamiento social humano de trabajar en conjunto y de forma armoniosa para realizar una acción común.

Sin embargo, si se considera que esta primera definición se pretende homologar a múltiples disciplinas relacionadas con la coordinación (como son la sociología, psicología, para este caso concreto el computo científico, etc.) el término puede resultar poco preciso o falto de contenido. Este hecho trae consigo la necesidad de proponer una definición de carácter formal (producto del estudio) y no intuitivo.

Cuando se recurre a la propia teoría de la coordinación o al resto de literatura relacionada con el tema, se observa que existen no una sino diversas definiciones sobre el mismo el término. A continuación se presenta una lista con algunas de estas definiciones y posteriormente un análisis sobre las mismas[3]:

> Procesamiento de información dentro de un sistema integrado por entidades de comunicación con diferentes estados de información.

> Esfuerzo conjunto de actores independientes para alcanzar una misma meta.

> Conjunto de actividades requeridas para manejar las dependencias dentro del flujo de trabajo.

> La coordinación es un tipo de clave dinámica que permite ligar tareas dentro de tareas mayores y con significados más completos. Además, ésta es un proceso íntimamente relacionado con la organización, dónde ésta última debe ser entendida como un requisito previo para que la coordinación pueda existir.[34].

> Coordinación es gestionar las dependencias entre actividades [48].

La lista anterior presenta cinco definiciones sobre el significado de *coordinación*. Como se puede observar algunas de éstas incluso parecen tener poca relación entre sí y muchas más definiciones distintas podrían adherirse a la lista.

En conclusión se pueden afirmar dos cosas: aun teniendo una idea intuitiva sobre lo que es *coordinación* rara vez se conoce todo lo que este proceso implica, y, definir el término *coordinación* no es una cuestión trivial y la precisión de su significado depende fuertemente del contexto en el que se presente.

2.4.2. Componentes de la coordinación

A continuación se propone una definición de carácter general sobre el significado del término *coordinación*[4]:

> **Acción** de realizar de forma armoniosa y por un conjunto de **actores** una o varias **actividades** para lograr una **meta** común.

[3] Las primeras tres definiciones de la lista han sido tomadas de [47].

[4] Esta definición se deriva del conjunto de definiciones propuestas anteriormente y del análisis posterior de las mismas.

COMPONENTE	ASOCIACIÓN
Metas	El objetivo o motivo de la coordinación
Actividades	El objetivo se desglosa en actividades
Actores	Las actividades se llevan acabo por actores
Interdependencias	Dependencias entre las actividades

Cuadro 2.4: Componentes del proceso de la coordinación.

Dentro de esta definición se destaca con tipografía más oscura una serie de elementos claves 2.4. Estos elementos, cuyas definiciones son expuestas a continuación, constituyen cada uno de los componentes que participan en el proceso de la coordinación [47]:

- **Actores.** En el proceso de coordinación intervienen un conjunto de actores.
- **Actividades.** Dentro del proceso de coordinación se realizan una o más actividades. El conjunto de actores que intervienen en el proceso realizan estas actividades para alcanzar algún objetivo común.
- **Metas.** Las metas son los objetivos comunes que se tienen dentro del proceso de coordinación. Éstas se alcanzan mediante la realización de actividades subyacentes al proceso.
- **Interdependencias.** La realización armoniosa de actividades delata el hecho de que éstas no son independientes unas de las otras. Las dependencias internas, o interdependencias, entre actividades representan las consideraciones que cada actividad debe tener, con respecto al resto, para lograr un objetivo de forma conjunta.

2.4.3. Coordinación dentro de los sistemas groupware

Desde una perspectiva tecnológica la realización conjunta de actividades independientes para alcanzar objetivos comunes conlleva, en algunas ocasiones, a escenarios de colaboración distribuidos. Sin embargo, esta distribución de actividades independientes tiene implicaciones que van más allá de su propia distribución en tiempo y espacio. Cada uno de los actores involucrados en un proceso de colaboración puede llegar a ser autónomo en términos de estrategias, heurísticas, perspectivas, conceptualizaciones, etc.

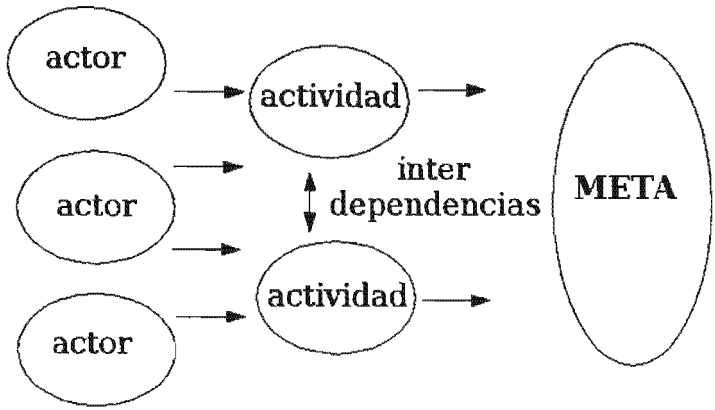


Figura 2.1: Relación entre los diferentes elementos que integran el proceso de coordinación.

Esta figura muestra la relación entre los diferentes elementos que integran el proceso de coordinación. El ejemplo describe un proceso de coordinación donde intervienen tres actores y dos actividades interdependientes que convergen en la misma meta. Las metas constituyen el objetivo del proceso en sí mismo. Las actividades son la descomposición de las metas en tareas que tienen una clara definición de qué actores y de qué forma éstos deben interactuar con cada una de ellas. Finalmente, las interdependencias representan las relaciones que existen entre las diversas actividades que constituyen el proceso de coordinación.

Por tal motivo, la complejidad del problema no radica sólo en soportar un conjunto de actividades distribuidas que pueden llegar a ser independientes entre sí. Sino en responder de qué forma las interdependencias que surgen entre estas actividades pueden ser encausadas a la solución de una meta común.

La coordinación, en principio, es un problema implícito al comportamiento social de los humanos que en la mayor parte de los casos es resuelto por éstos de forma tácita. Sin embargo, trasladar el problema de la coordinación a los sistemas groupware lo convierte en un problema complejo que actualmente no presenta soluciones definitivas.

En términos computacionales, para un sistema groupware coordinar significa tener la capacidad de encausar las interacciones que acontecen en su espacio de trabajo compartido hacia la solución de una meta común. Donde tales interacciones son el resultado de actividades independientes que se han originado en sistemas de computo distribuidos.

Lo anterior es equivalente a responder de forma lógica a las preguntas[5]:

- ¿quién hace algo?
- ¿qué hace?
- ¿sobre qué lo hace?
- ¿dónde lo hace?
- ¿cómo?
- ¿en respuesta a qué?
- ¿bajo qué restricciones?

Dónde la respuesta particular a cada una de estas cuestiones dependerá de los requerimientos de cada sistema groupware.

Importancia de la coordinación en los sistemas groupware.

Dentro del dominio CSCW/CSCL una de las líneas de investigación más importantes y estudiadas es la coordinación. Entender de que forma las computadoras pueden constituirse en instrumentos eficientes capaces de reducir la complejidad de coordinar actividades cooperativas, interdependientes e individualmente conducidas es una prioridad en los sistemas CSCW/CSCL. La importancia de proporcionar el soporte adecuado de coordinación (proveer soporte de coordinación inexorablemente implica proveer además soporte de conciencia de grupo) para cada sistema CSCW/CSCL se convierte en una cuestión fundamental y determinante, incluso para la propia utilización de éste [6].

Como resultado de proveer mejores mecanismos de coordinación dentro de los sistemas groupware, en los últimos años han aparecido un gran número de herramientas como son modelos conceptuales [21], marcos teóricos[48], lenguajes de modelado [94], lenguajes interpretados [44][45], patrones [67], monoaplicaciones CSCW, marcos para desarrollo con soportes especializados [67][19].

De la lista anterior se destaca la utilización de marcos como una tendencia vigente, y probablemente la más aceptada, para el desarrollo de sistemas groupware.

[5]Los primeros cuatro elementos de la lista representan directamente la información relacionada con la conciencia del espacio de trabajo. Se destaca la relevancia de esta información dentro del proceso de coordinación de los sistemas groupware.

[6]En [80], por ejemplo, se listan varias herramientas groupware que no han logrado trascender a causa del soporte de coordinación que proporcionaban. En algunos casos se tenía un soporte demasiado rígido y en otros el soporte es demasiado complejo.

Los marcos en los últimos años han realizado un giro tecnológico en torno a la incorporación de componentes de software. La fusión tecnológica Marcos/CBSE está ofreciendo resultados positivos en diversas áreas de la computación distribuida y por tal motivo se ha convertido en una apuesta prometedora para el caso de los sistemas groupware. A continuación se presenta una perspectiva general de cada una de éstas tecnologías.

2.5. Ingeniería de Software Basada en Componentes

La Ingeniería de Software Basada en Componentes constituye una extensión natural de la orientación a objetos para los entornos abiertos que pretende el desarrollo y utilización de componentes reutilizables de forma generalizada.

Clemens Szyperski [92].

2.5.1. Ingeniería de Software e Ingeniería de Software Basada en Componentes

Los constantes y vertiginosos avances que en los últimos años han tenido los sistemas de cómputo han derivado en una lista de necesidades que hasta hace poco no se tenían. Particularmente, en materia de cómputo distribuido, los sistemas han aumentado notablemente su complejidad de desarrollo.

Paradigmas como la Programación Orientada a Objetos que hasta hace sólo unos años constituían un sustento sólido para el desarrollo de sistemas distribuidos, que se caracterizaban por ser cerrados y con una necesidad de interoperabilidad mínima, se han visto irremediablemente rebasados por las nuevas necesidades que afronta este dominio.

Sistemas distribuidos, abiertos[7], y, preparados para coexistir en ambientes altamente dinámicos con otros sistemas, son cada día más necesarios para ofrecer soluciones verdaderamente integrales. Ante este hecho, la incapacidad de dar solución a este giro que han dado los sistemas de computo, ha surgido un nuevo paradigma para el desarrollo de software: Ingeniería de Software Basada en Componentes (CBSE).

[7]Un sistema abierto es un sistema independientemente extensible que permite a componentes heterogéneos abandonar o ingresar al sistema de forma dinámica. Independientemente extensible significa que un sistema que puede ser dinámicamente extendido. Por lo tanto estos sistemas son inherentemente evolutivos ya que la vida de sus componentes es mas corta que la del propio sistema.

La motivación de dejar atrás piezas enormes y monolíticas de software para descomponer un sistema en piezas o fracciones binarias que permitan el desarrollos modularizados, abiertos y altamente reutilizables ha cautivando la atención de la industria y la investigación de todo el mundo. A lo largo de la última década algunos autores han llamado a la Ingeniería de Software Basada en Componentes la tecnología que revolucionará el mundo del software:

> La Ingeniería de Software Basada en Componentes es una tecnología que está por sacudir a la industria [89].

2.5.2. Conceptos básicos

Componente

A pesar de que la palabra componente es un concepto largamente utilizado dentro de la Ingeniería de software definir el significado de éste dentro del ámbito de la propia CBSE es una tarea complicada. En los últimos años diversas definiciones han aparecido y se han sumado a otro tanto ya existente. A continuación, como prueba de este hecho, se muestra una lista con algunas de las definiciones más aceptadas sobre este término:

Definiciones del término componente:

- Un componente de software es un módulo lógicamente cohesivo y con poco acoplamiento que denota una sola abstracción [4].
- Un componente de software es una abstracción estática con conectores [58].
- Los objetos distribuidos son, por definición, componentes [60].
- Componentes reutilizables de software son elementos con un autocontenido, claramente identificables que describen y/o realizan funciones específicas, tienen interfaces claras, documentación apropiada y un propósito de reutilización bien definido [78].
- Programación orientada a componentes: polimorfismo + liga realmente pospuesta + encapsulamiento real y reforzado + herencia de interfaces + reuso binario [5].
- Un componente es una unidad funcional de un sistema; puede ir desde una función de bajo nivel hasta una de alto nivel; puede ser atómica o compuesta; es reutilizable o compartible. El modelo de componentes difiere del de objetos; éste último no contiene todos los elementos necesarios para describir los componentes y su ensamble. Atributos esenciales de componentes: independencia del contexto (se conectan arbitrariamente), transparencia de localidad, conexión heterogénea [6].

- Un componente, en general, es un paquete coherente de artefactos de software que pueden ser desarrollados independientemente y entregados como unidad que puede ser compuesta con otras, sin necesidad de cambios, para formar algo mayor. Un componente, en código, es un paquete coherente de software que: puede ser desarrollado y entregado independientemente, tiene interfaces explícitas y bien especificadas para servicios ofrecidos, tiene interfaces explícitas y bien especificadas para servicios requeridos, puede componerse con otros componentes, tal vez adaptando propiedades, pero sin modificación [18].

- Los componentes de software son bloques reutilizables de construcción de sistemas de software. Encapsulan aplicaciones o servicios técnicos con sentido semántico. Difieren de otros tipos de módulos reutilizables en que pueden modificarse en tiempo de diseño o de ejecución. Los componentes restringen acceso vía una o más interfaces públicas que definen propiedades, métodos y eventos que permiten comunicación. Los componentes existen y operan en contenedores que les brindan contexto compartido para interactuar con otros componentes y acceso a servicios del sistema [42].

- Los componentes son las menores unidades auto administradas, independientes y útiles de un sistema que trabaja en ambientes múltiples. Pueden ser un objeto, pero no es indispensable [43].

- Un componente es una parte de un sistema, pero a diferencia de módulos y otros elementos, es como una pequeña aplicación que viene lista para usarse. Para que funcione se requiere una herramienta que especifique la manera de crear componentes y armar aplicaciones [71].

- Componente de software es código que implementa un conjunto de interfaces bien definidas. Es un segmento manejable de lógica. No es una aplicación completa, no corre aisladamente sino como parte de un conjunto [72].

- Un componente de software es una unidad de composición que únicamente especifica, mediante un contrato, las interfaces y las dependencias explícitas del contexto. Un componente de software puede desplegarse independientemente y es sujeto de composición por terceros [89].

Como se puede observar en la lista anterior existen diversas definiciones; muchas de ellas incluso no coincidentes entre sí. A partir de este hecho se puede inferir que definir el concepto de componente es un tema ríspido que puede derivar con facilidad en confusiones o conceptos equívocos. El término componente puede tener matices distintos de acuerdo al propio ámbito de cada definición.

Por ejemplo, una definición del término componente desde la perspectiva del paradigma de orientación a objetos puede variar de otra hecha desde la perspectiva de CBSE.

Una de las definiciones que ha causado mayor impacto dentro de la literatura de CBSE es la de *Clemens Szyperski*. Por tal motivo, y para los fines de este trabajo, se ha decidido adoptar como propia tal definición.

Según *Clemens Szyperski* [89] un componente de software es:

> Una unidad de composición que únicamente especifica, mediante un contrato, las interfaces y las dependencias explícitas del contexto. Un componente de software puede desplegarse independientemente y es sujeto dè composición por terceros.

A partir de esta definición un componente podría ser entendido como la unidad de construcción básica de un sistema de software basado en componentes. Donde todo el conjunto de unidades representan las piezas lógicas que integran una aplicación[8]. Cada una de las piezas especifica a través de sus interfaces que provee y que recibe.

A continuación se definen algunos términos relacionados con esta definición de componente:

Interfaces

Las interfaces de un componente pueden ser entendidas como los puntos de acceso a un componente. Estos puntos de acceso permiten a clientes, que normalmente estarán representados por otros componentes, tener acceso a los servicios que este componente ofrece. Un componente puede tener múltiples puntos de acceso, múltiples interfaces a través de las cuales los clientes accederán a sus múltiples servicios.

Contratos

Un contrato debe ser entendido como la base de acuerdo entre un componente y sus clientes. Si se considera que los componentes son sujeto de composición por terceras partes y que éstas no tienen ningún conocimiento previo sobre el componente el contrato representa la única forma de interactuar. A través de éste los clientes conocen de que forma deben acceder al componente para obtener un determinado servicio.

[8]Se puede pensar de forma análoga en las piezas que integran un rompecabezas.

Dependencias del contexto

Las dependencias del contexto de un componente son una descripción de los elementos que el propio entorno debe proporcionar al componente para su buen funcionamiento. Por ejemplo, si se piensa en un componente que muestra el contenido de un archivo una dependencia clara del contexto está dada por el propio archivo.

2.5.3. Diferencias entre objetos, clases y componentes.

Existe una gran confusión entre las diferencias que existen entre los conceptos de un objeto y un componente. Es importante aclarar que al menos en teoría ambos conceptos son completamente ortogonales [90]. A pesar de que la CBSE nace como una extensión de la orientación a objetos es posible desarrollar CBSE sin utilizar técnicas de orientación a objetos.

Los componentes se caracterizan por ser unidades de composición y elementos de despliegue independientes. Por tal motivo un componente no puede ser desplegado de forma parcial.

El despliegue de componentes debe ser entendido como un proceso complejo en el que, normalmente, intervienen técnicas de introspección y generación de código. Si un componente contiene clases el proceso de despliegue las prepara para la creación de sus instancias. Las instancias, entonces, son instancias de clases y no de componentes. Estas instancias de las clases son los objetos. Sin embargo, existen algunos autores como [46] que definen un objeto como la instancia de un componente.

Uno de los requerimientos más característicos de un componente es que éste debe estar separado de su entorno y del resto de los componentes. Además, un componente debe especificar claramente qué provee y qué requiere. Dicho de otra forma, un componente necesita encapsular su implementación e interactuar con su entorno a través de interfaces. Típicamente, en la práctica, un componente tomará vida a través de objetos. Por lo que este componente deberá contener una o más clases. Sin embargo, no es una necesidad que un componente contenga alguna clase o solamente clases. Un componente podría contener procedimientos tradicionales no encapsulados dentro de objetos.

En el caso de un objeto éste se caracteriza por ser una unidad de instanciación. Al ser una instancia un objeto no puede ser instanciado parcialmente. Los objetos son instancias de clases. Entonces, si una clase, un conjunto de reglas que describe como debe crearse un objeto en memoria, se instancia da lugar a un bloque contiguo de celdas de memoria que son llamadas objetos o instancias de clases.

En realidad el concepto de clase se asemeja más al de componente que al de objeto. Un componente y una clase, o un conjunto de éstas[9], son elementos estáticos cuya existencia está más relacionada con un despliegue que con una instanciación. Sin embargo, en ningún caso deben entenderse estos conceptos como sinónimos. Mientras un componente puede contener múltiples clases, una clase, necesariamente, debe estar contenida en un solo componente.

2.5.4. Plataformas para el desarrollo de componentes

Existen fundamentalmente tres propuestas que dominan el mercado para el desarrollo de componentes de software.

- Sun Microsystems.
- Microsoft.
- Object Management Group.

Sun Microsystems

La plataforma de componentes de *Sun Microsystems* es J2EE (*Java 2 Platform Enterprise Edition*) [85]. J2EE en realidad no es una aplicación sino un estándar para el desarrollo de aplicaciones *enterprise*[10].

Actualmente existen diversas soluciones que ofrecen *middleware*[11] para componentes desarrollados bajo este estándar. J2EE está adoptado por más de más de treinta empresas diferentes de software. Esto permite tener diversas opciones en la selección de productos relacionados con el desarrollo de componentes. El modelo de componentes *enterprise* que utiliza J2EE son los EJB (*Enterprise Java Beans*)[84].

[9] En la literatura relacionada con CBSE un conjunto de clases es denominado *modulo*.

[10] El término *enterprise* es utilizado para referirse a aplicaciones distribuidas de gran escala.

[11] El término *middleware* hace referencia a una capa de software intermedia que es proveedora de servicios. En el caso de sistemas distribuidos el *middlware* normalmente está formado por servicios como: manejo de operaciones transaccionales, envío de datos seguro, balanceo de carga de trabajo, etc.

Otras tecnologías relacionadas con el desarrollo de EJB son Java y Javabeans:

Java. Es un lenguaje de programación que soporta diversas técnicas de la programación orientada a objetos. La principal ventaja de Java es que su código es portable a diversas arquitecturas de hardware y sistemas operativos. Para conseguir esta portabilidad Java debe ser compilado a un código intermedio conocido como *bytecode*. Posteriormente este nuevo código es interpretado dentro de un ambiente de ejecución conocido como JRE *(Java Runtime Environment)*[27].

Javabeans. Esta es otra especificación a través de la cual es posible desarrollar componentes de software reutilizables [87]. Estos componentes también llevan por nombre Javabeans. Entonces, Javabeans es, por un lado, una especificación para el desarrollo de componentes de software, y por el otro, los propios componentes desarrollados bajo esta misma especificación. El modelo de componentes de los Javabeans, a diferencia de los EJB, no contempla la utilización de *middleware* de forma implícita debido a que los Javabeans no son componentes necesariamente distribuidos.

Microsoft

Microsoft.Net es una familia de productos que ofrece, entre otras cosas, una plataforma para el desarrollo de componentes de software. Microsoft.Net en realidad es sólo la nueva versión de la plataforma *Microsoft DNA (Distributed interNet Applications)*.

La plataforma de desarrollo y ejecución de aplicaciones de Microsoft.Net es *.Net Framework*. Esta plataforma está compuesta por dos elementos básicos: un conjunto de bibliotecas que facilita el desarrollo de aplicaciones de software (basadas o no en componentes), y, un ambiente de ejecución conocido como CLR (*Common Language Runtime*) [56].

Una de las características más destacadas de esta plataforma es la capacidad de ofrecer interoperabilidad entre componentes desarrollados en diferentes lenguajes. Esto se logra a través de la utilización de un lenguaje intermedio, conocido como IL (*Intermediate Lenguage*), al que son traducidos todos los programas independientemente del lenguaje en el que han sido desarrollados. Además, CLR cuenta con un compilador JIT (*Just In Time*)[12] que es el encargado de compilar el código IL a código nativo del sistema y posteriormente ejecutarlo [13].

El modelo de componentes de Microsoft es *.Net Managed Componentes*. Este modelo, al igual que los EJB, cuenta con middleware para el desarrollo de aplicaciones *enterprise*.

[12]Los compiladores JIT compilan y ejecutan el código en un mismo proceso.

En realidad el modelo de componentes de Microsoft.Net subyace sobre sus modelo anteriores: COM (*Component Object Model*) y COM+ (*Component Object Model Plus*).

COM más que un modelo de componentes es una normativa de Microsoft para el desarrollo de clases de software reutilizables. Los objetos COM en realidad son equivalentes a clases comunes que cumplen con convenciones establecidas *a priori*. Lo que les permite ser reutilizados por otros programas sin necesidad de ninguna adecuación. COM+ es la incorporación de middleware (*COM+ Component Services*) a los objetos COM.

Algunas desventajas de esta plataforma es que no es portable, sólo funciona bajo sistemas operativos propietarios de la misma compañía, además, toda la gama de productos relacionados con el desarrollo de componentes cuentan con Microsoft como único proveedor.

Object Management Group

Object Management Group (OMG) es un consorcio formado por diversas empresas relacionadas con el desarrollo de software. Uno de los objetivos principales de OMG es crear una especificación estándar para arquitecturas de software distribuidas: *Object Management Arquitecture* (OMA).

A partir de esta iniciativa OMG ha propuesto una especificación que es parte fundamental de OMA: CORBA (*Common Object Request Broker Architecture*) [59]. CORBA es una especificación para lograr interoperabilidad entre aplicaciones independientemente de la plataforma, sistema operativo o el lenguaje de programación en que fueron desarrolladas [69].

El modelo de componentes de OMG (*Object Management Group*) es conocido como *CORBA Component Model* (CCM). En realidad CCM es una especificación complementaria de la especificación CORBA 3.0 [2]. Los componentes CCM son sólo una extensión de objetos CORBA. A través de esta extensión los objetos CORBA pueden acceder de forma implícita a una serie de servicios (middleware) conocidos como *CORBA Services*: servicio de operaciones transaccionales, de transferencia segura de datos, de persistencia de datos y de notificación de eventos.

2.6. Marcos para desarrollo de aplicaciones

> *Un Marco es un diseño reutilizable, de todo o parte de un sistema, que está representado por un conjunto de clases abstractas (software) y la forma en que los ejemplares de éstas actúan (patrones).*
>
> *Ralph E. Johnson* [38].

2.6.1. Marcos y sus elementos

Los *Marcos o Frameworks* son una técnica para la reutilización de software que aprovecha tres características fundamentales del paradigma de programación orientada o objetos[13]:

- Abstracción.
- Polimorfismo.
- Herencia.

Técnicamente, un marco es conjunto de clases abstractas y un diseño que especifica de que forma interactúan estás clases.

Una clase abstracta, en programación orientada a objetos, es una clase que no puede ser ejemplarizada [35]. Normalmente representan objetos reales y abstractos cuyos ejemplares no existen físicamente. Estas clases sólo fungen como una plantilla que denota comportamiento y ejemplarización para otras clases. Por lo tanto, las clases abstractas son *superclases* o clases padre que transmiten información de forma hereditaria a sus *subclases* o clases hijas. En un marco, las clases abstractas contienen el código que deberá ser reutilizado por los componentes[14] de ese marco.

El diseño está dado por uno o más patrones que le permiten al marco homologar el comportamiento de sus componentes. Los componentes deben adecuarse a ese diseño y es la manera en que el marco sabe *a priori* como debe tratarlos. Una parte de este comportamiento también está predefinido dentro de las clases abstractas.

La reutilización de un marco tiene dos vertientes fundamentales: una reutilización a nivel de código (a diseño de bajo nivel) y una reutilización a nivel de diseño (entiéndase diseño de alto nivel). El diseño de alto nivel describe el comportamiento de todos los elementos de un sistema mientras que el código permite reutilizar el código que es común a todos esos elementos. Por tal motivo un marco no es una entidad de software que pueda tener un uso discrecional. Un marco sólo puede utilizarse dentro de un conjunto cerrado de aplicaciones. Aplicaciones que comparten parte de su diseño y de su implementación. Aplicaciones que deberán cumplir con una serie de normativas impuestas por el propio marco: un diseño y un modelo específico de colaboración.

[13]Por lo tanto, los marcos son una técnica de reutilización utilizada en la programación orientada a objetos.

[14]Es importante resaltar que el término componente dentro del ámbito de los marcos hace referencia a las aplicaciones que se construyen a partir del marco. Estas aplicaciones constituyen los componentes del marco y no tienen que estar realizadas necesariamente bajo el paradigma de CBSE.

En resumen, se podrían definir los marcos de dos formas: el esqueleto de una aplicación que puede ser adaptado y reutilizado por los programadores [38], o, la arquitectura de un sistema, las clases y/o componentes que hay en él y la forma en que éstos interactúan [37].

2.6.2. Marcos y componentes

El concepto de marco está íntimamente relacionado con el objetivo de reutilización de software. Por ello el concepto de componente dentro de los marcos es muy aludido. Un componente hace referencia a aquellas piezas de software que pueden ser conectadas de forma simple para la creación de un nuevo sistema. Estás piezas se construyen a través de la utilización del marco.

Esta definición podría llevar al planteamiento de si un marco es o no un componente. Un marco podría ser un componente sólo en el sentido estricto en que una misma aplicación, en principio, puede recurrir a la reutilización de diversos marcos. En realidad las aplicaciones sí son independientes de los marcos. Sin embargo, un marco no puede entenderse como un componente al que los programadores pueden recurrir a través de sus interfaces ignorando todas sus consideraciones internas (un sistema de caja negra).

El conjunto de interfaces, o los puntos de acceso a un marco, a diferencia de lo que debería ser un componente, es complejo; los programadores deben entender mediante un proceso no trivial de que forma interactuar con un marco.

La verdadera relación Marco/Componente puede ser vista desde dos perspectivas distintas:

1. Un marco en un proveedor de servicios. Una de las funciones primarias de un marco es proveer a un conjunto de componentes de servicios.

2. Un entorno para el desarrollo de componentes. La segunda función de un marco es ofrecer a los programadores de un entorno para el desarrollo de nuevos componentes.

Los servicios que ofrece un marco a los componentes pueden variar de acuerdo a la naturaleza de los propios componentes. Por ejemplo, algunos de los servicios básicos que usualmente ofrece un marco son: manejo de excepciones, intercambio de datos, invocación de operaciones, entre otras; sin embargo, un marco para el desarrollo de aplicaciones distribuidas podría ofrecer servicios más especializados del dominio como: balanceo de carga, transacciones, operaciones concurrentes, entre otras.

DESVENTAJAS DE LOS MARCOS
Difíciles de utilizar
Difíciles de desarrollar
Dependen de un lenguaje orientado a objetos

Cuadro 2.5: Desventajas más importantes de los marcos o *frameworks*.

2.6.3. Marcos y patrones

Los patrones son una técnica de reutilización de software centrada en diseño. Los patrones normalmente describen un problema recurrente y una solución de diseño a este problema (así como los escenarios donde la solución es válida).

Los marcos son distintos de los patrones porque constituyen más que sólo diseño; son también código. Esta reutilización de código permite a los programadores construir aplicaciones de forma ágil. En realidad, un marco contiene uno o varios patrones de diseño. Dicho de otra forma, los patrones son elementos contenidos en los marcos aunque no son los únicos.

Mientras la funcionalidad de un patrón debe ser ilustrada a través de una aplicación un marco en sí mismo es una aplicación. Sin embargo, más que una aplicación, los marcos deben ser entendidos como *meta aplicaciones*: aplicaciones, que utilizan patrones de diseño y software, para generar otras aplicaciones.

2.6.4. Desventajas de los marcos

A continuación se listan las desventajas más importantes de los marcos (ver cuadro 2.5):

- Difíciles de utilizar. La principal desventaja de los marcos es que son herramientas difíciles de aprender. Documentación y el entrenamiento de los programadores es necesario para una utilización correcta.
- Difíciles de desarrollar. El desarrollo de un marco es también es una tarea compleja. El grado de dificultad de diseñar y programar un marco excede considerablemente el de una aplicación convencional.
- Dependencia de un lenguaje orientado a objetos. Los marcos deben ser desarrollados con un lenguaje que permita utilizar técnicas de orientación a objetos y, posteriormente, las aplicaciones o componentes que se integren a éste tendrán una dependencia funcional de ese mismo lenguaje[15].

[15]Existen algunas alternativas, como CORBA [59], para lograr la interoperabilidad entre lenguajes, sin embargo, éstas aumentan considerablemente la dificultad de desarrollo y de utilización de un marco.

2.6.5. Marcos CSCW/CSCL

El dominio de los sistemas CSCW/CSCL, de igual forma que cualquier otro dominio relacionado con la ingeniería de software, se ha visto fuertemente influenciado por las tendencias dominantes para el desarrollo de software en cada momento.

Así, en su primera etapa los sistemas CSCW/CSCL han intentado proveer soluciones monolíticas. Sistemas de software enormes que contenían embebida la lógica propia de la colaboración además de la de la aplicación. Sin embargo, las soluciones eran costosas de desarrollar y difíciles de mantener (tanto la lógica de colaboración como la lógica de la aplicación) lo que tenía como consecuencia sistemas donde la reutilización de código era ínfima.

La segunda generación de sistemas CSCW/CSCL se distinguió entre otras cosas por importantes aportaciones para la reutilización de código. Estos sistemas aportaron bibliotecas [76][77][86][65] y, posteriormente, también se aportaron marcos [28][9][26] que facilitaban el desarrollo y mejoraban notablemente la reutilización dentro de los sistemas CSCW/CSCL.

Esta última solución, la utilización de marcos para el desarrollo de aplicaciones CSCW/CSCL, se mantiene como una tendencia dominante que recientemente se ha fortificado con la aparición de CBSE (un paradigma de programación centrado en la reutilización de software). La fusión de ambas tecnologías, marcos y componentes, ha tenido ya aportaciones importantes en diversos dominios de la computación distribuida [55][87] y dentro de los sistemas CSCW/CSCL, particularmente, parece ser ya una solución concensada al problema de reutilización y simplificación.

2.6.6. Problemas de los Marcos CSCW/CSCL

Uno de los problemas principales con el desarrollo de sistemas CSCW/CSCL mediante la utilización de marcos es que éstos aportan notables mejoras en la reutilización de servicios como: objetos compartidos, soporte a roles, sesiones, etc. sin embargo, el soporte que proveen a otros servicios como: coordinación y conciencia de grupo, debido a su complejidad, suele ser pobre y no resulta ni mínimamente suficiente para proveer un verdadero espacio de trabajo compartido de forma virtual.

Existen otras soluciones como [44][19] que pretenden abordar el problema de proveer mecanismos de coordinación de forma independiente. Marcos especializados en proveer servicios de coordinación. Sin embargo al ser soluciones aisladas que no integran ambos conceptos la solución que ofrecen tampoco parece la mejor. La conciencia de grupo es siempre requerida para coordinar las actividades de un grupo, independientemente de la tarea que se realice [16][31].

En realidad, proveer verdaderos servicios de coordinación en los sistemas groupware requiere de una integración de éstos con servicios de conciencia de grupo. La conciencia de grupo es una parte importante de cualquier sistema groupware síncrono [33] y un elemento clave para la coordinación de actividades en un sistema groupware. No se puede efectuar una coordinación efectiva si ésta no se nutre en todo momento de las acciones que acontecen en un espacio de trabajo compartido.

En resumen, en este trabajo expone la necesidad de proveer a través de marcos o bibliotecas mecanismos para desarrollar de forma más simple servicios de conciencia de grupo y coordinación que puedan ser reutilizables en sistemas groupware de propósito general (CSCW/CSCL).

2.7. Discusión

A lo largo de este capítulo se han expuesto las tendencias actuales en el desarrollo de aplicaciones CSCW/CSCL. El desarrollo de estas aplicaciones a través de la fusión tecnológica Marcos/Componentes parece ser una solución idónea al problema de la reutilización y del desarrollo simplificado de aplicaciones groupware. Sin embargo, y como se ha señalado previamente, un problema vigente es la incorporación de servicios robustos de coordinación y conciencia de grupo en marcos CSCW/CSCL.

La complejidad de desarrollar y mantener servicios de coordinación y conciencia de grupo dentro de sistemas CSCW/CSCL es alta, sin embargo, la importancia de éstos elementos dentro del proceso de colaboración hace inexorable su inclusión. Por tal motivo, una necesidad primaria en la evolución de los sistemas CSCW/CSCL es simplificar el desarrollo y mantenimiento de estos servicios a través su inclusión en marcos CSCW/CSCL.

El siguiente capítulo, precisamente, detalla la creación de un marco conceptual para la generación de modelos de coordinación y conciencia de grupo en sistemas CSCW/CSCL. Este marco conceptual pretende sentar las bases para el posterior diseño y desarrollo de un marco CSCW/CSCL, basado en componentes de software, que simplifique el desarrollo y mantenimiento de aplicaciones a través de la inclusión de servicios robustos de coordinación y conciencia de grupo.

Capítulo 3

Marco conceptual para un modelo de coordinación y conciencia de grupo en sistemas CSCW/CSCL

3.1. Conceptos relacionados con un modelo de coordinación

Para exponer algunos conceptos relacionados con la coordinación dentro de los sistemas groupware se propone el siguiente ejemplo:

Rompecabezas colaborativo síncrono

Se tienen varios usuarios que desde diferentes lugares físicos, a través de sus computadoras, colaboran durante el mismo espacio de tiempo en la solución de un rompecabezas. Para convertirse en actores del escenario de colaboración los usuarios de esta aplicación (rompecabezas) deben adquirir uno de dos posibles roles: alumno/jugador y/o profesor/revisor.

Debe considerarse, en el caso de los actores, que éstos podrían estar determinados por un solo usuario, un grupo de usuarios o incluso por un proceso de computo. Por ejemplo, cada jugador podría estar constituido por un grupo de usuarios y cada profesor por una rutina de computo que revisa las soluciones propuestas.

Los actores que tienen el rol *alumno* estarán a cargo de resolver el rompecabezas (actividad uno), mientras que los actores que tienen el rol *profesor* deberán revisar si las soluciones propuestas por los alumnos son correctas (actividad dos). La segunda actividad entra en vigor si y sólo si la primera actividad ha concluido.

Los objetos compartidos son las piezas del rompecabezas por lo que los jugadores interactúan unos con otros a través de éstas. Las acciones permisibles (determinadas por los propios objetos compartidos) son: ver pieza, colocar pieza y recolocar pieza. Debe pensarse que se trata de un sistema distribuido donde los usuarios no comparten necesariamente la misma vista del modelo (lo que da sentido a tener una acción que permite ver la pieza).

El objetivo de la colaboración es armar correctamente el rompecabezas a través del procedimiento resolver/revisar rompecabezas.

3.1.1. Coordinación de acciones y coordinación de actividades.

En los sistemas groupware la coordinación se da básicamente a dos niveles:

- Nivel de actividad. Un modelo de coordinación a nivel de actividad constituye la secuencia valida de actividades que conforman un proceso.
- Nivel de objeto. A nivel de objeto la coordinación se encarga de gestionar las accesos secuenciales y simultáneos a los mismos objetos.

Para el ejemplo del rompecabezas colaborativo la decisión sobre la forma de reaccionar ante dos jugadores que solicitan colocar la misma ficha concierne a una coordinación a nivel de acciones. Por otro lado, la decisión sobre las actividades (la actividad resolver el rompecabezas es necesaria para iniciar la actividad revisión de la solución) concierne a una coordinación a nivel de actividad.

La coordinación a nivel de objeto usualmente recurre a soluciones de tipo primario como candados. Este principio consiste en bloquear un objeto cuando un participante está modificándolo. De esta manera se evita que otro participante modifique simultáneamente el mismo objeto. Este principio es típicamente utilizado dentro de las bases de datos y la distribución de procesos para evitar la inconsistencia de información. Sin embargo, en el caso de los sistemas groupware esta solución no resulta suficiente y resulta un reto proveer mecanismos de coordinación más flexibles que los candados [21], como los propuestos por este trabajo de tesis.

3.1.2. Actividad

El modelo de coordinación dentro de un sistema groupware describe las actividades que cada participante puede realizar y la forma en que estas actividades deben ser coordinadas para que el grupo logre su objetivo [21].

Para esta definición el concepto de actividad es un elemento clave ya que las dos tareas de un modelo de coordinación están descritas a partir del concepto actividad: ¿qué actividades puede realizar cada participante? y ¿de qué forma estas actividades deben ser realizadas?

El concepto de actividad se define de la siguiente forma:

> Una actividad es un conjunto potencial de acciones que un actor puede realizar sobre los objetos disponibles [21].

En el ejemplo del rompecabezas colaborativo las actividades han sido definidas explícitamente como: resolver rompecabezas y revisar rompecabezas. En el caso de la actividad resolver rompecabezas el conjunto potencial de acciones permisibles son ver pieza, colocar pieza y recolocar pieza, mientras que en el caso de la actividad revisar rompecabezas la única acción permisible es revisar pieza. En ambas actividades las acciones se realizan sobre los objetos compartidos pieza.

3.1.3. Acción e Interacción

Acción

Dentro de una misma actividad se pueden tener acciones concurrentes o secuenciales a un mismo objeto, la coordinación a nivel de acciones se encarga precisamente de gestionar esa concurrencia. Por tal motivo el concepto de acción tiene una gran relevancia dentro de la coordinación. Pero no sólo el concepto de acción sino toda la información adicional que rodea a este concepto: el objeto sobre el que se realiza y el actor que la causa.

Una acción se pude definir como:

> El ejercicio de la posibilidad de hacer algo [70].

Aunque esta definición es de carácter general y podría parecer demasiado simplista extrapolarla al ámbito de los sistemas groupware es posible matizar su contenido de la siguiente forma: El ejercicio, de un actor, de realizar las operaciones permitidas sobre los objetos compartidos del sistema.

En el caso del rompecabezas colaborativo un ejemplo de operación permisible podría ser colocar una pieza. Donde el rango de aplicación estaría delimitado por el propio objeto compartido (la pieza del rompecabezas que se pretende colocar).

Interacción

Existen diferentes tipos de acciones y algunas de sus diversas clasificaciones tienen origen en las causas o consecuencias que conlleva cada una. Un tipo de acción que resulta particularmente interesante por su importancia dentro de los escenarios groupware son las interacciones.

El concepto de interacción podría definirse como sigue:

> Acción que se ejerce recíprocamente entre dos o más objetos, agentes, fuerzas, funciones, etc. [70].

Nuevamente, por tratarse de una definición de carácter general, el contenido de ésta resulta insuficiente para llevarse al ámbito de los sistemas groupware. Para este caso en particular, una *interacción* representa, entre otras cosas, una acción cuya percepción o consecuencia va más allá de su emisor. Lo anterior puede tener dos interpretaciones válidas. La acción, al tener consecuencias que van más allá de su emisor, se convierte en una interacción o, la interacción es el producto resultante de una acción cuya percepción ha ido más allá de su emisor. En ambos casos, el concepto de interacción implica, necesariamente, la existencia de una acción que provoque reacciones o consecuencias en el escenario de colaboración (ésta es la reciprocidad de la que se habla en la primera definición).

3.1.4. Otros conceptos

Procedimiento

Un procedimiento es un conjunto de actividades ordenadas. Por lo tanto, en el caso del ejemplo del rompecabezas, un procedimiento sería realizar las actividades resolver/revisar rompecabezas en ese mismo orden. Normalmente los sistemas groupware están diseñados con sólo un procedimiento, sin embargo, esto depende de las necesidades particulares de cada uno. Existen algunos sistemas groupware llamados de flujo de trabajo (*workflow*) que usualmente sí consideran diversos procedimientos.

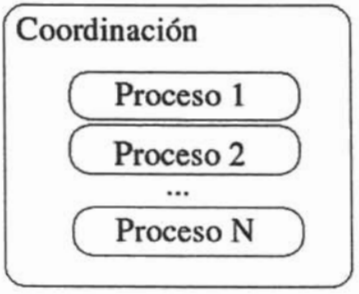


Figura 3.1: Capas de la coordinación.

Tarea

Dentro de un sistema groupware los ejemplares que se crean de cada actividad son llamados tarea. Una misma actividad puede tener diversos ejemplares (tareas) dentro de la misma aplicación groupware.

El conjunto de tareas que representan las instancias de una misma actividad se conocen con el nombre de *endeavor*. Por lo tanto un *endeavor* constituye la instancia de un procedimiento.

Tarea inactiva. Una tarea está inactiva si no ha sido iniciada o si ha sido finalizada. En cualquier otro caso se trata de una *Tarea activa.* Cuando una tarea (la instancia de una actividad) está activa significa que ha sido iniciada pero no ha sido concluida. En consecuencia las operaciones y objetos definidos para esta tarea están disponibles para el actor correspondiente.

3.2. Dependencias subyacentes al modelo

Así como existe la necesidad de coordinar para colaborar para coordinar también existen otras necesidades. La coordinación y sus necesidades particulares podrían entenderse de dos formas:

- La coordinación es un meta proceso, un proceso formado por una secuencia de procesos donde las necesidades de cada proceso se representan por la antecesión de otro proceso (Figura 3.2).
- Un modelo de capas. La coordinación está constituida por capas donde la primera capa subyace sobre otras pero se nutre sólo de la capa baja subsecuente. A su vez esta segunda capa se nutre sólo de su capa baja subsecuente y así sucesivamente (Figura 3.1).

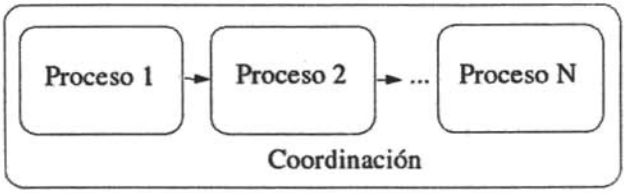


Figura 3.2: Procesos de la coordinación.

PROCESOS
Coordinación
Herramientas para la toma de decisiones de grupo
Canales de comunicación
Protocolos comunes para intercambio de información

Cuadro 3.1: Marco conceptual de la coordinación y sus procesos subyacentes.

En [48] se propone un marco conceptual (Cuadro 3.1) donde se identifica a través de un modelo de capas las dependencias más fuertes del proceso de coordinación. Este marco se sustenta en la teoría de la coordinación y su objetivo es proveer una herramienta de análisis para asistir el diseño de sistemas para el trabajo colaborativo.

3.2.1. Herramientas para la toma de decisiones de grupo

Colaborar representa un compromiso colectivo (de un grupo de actores) de participación en un esfuerzo coordinado para resolver un problema de forma conjunta [74]. Hablar de participación colectiva hace evidente la existencia de un grupo de actores en este proceso. Y hablar de coordinación demuestra la necesidad de resolver las dependencias internas que existen entre los esfuerzos o acciones que se gestan. Si no existieran estas dependencias tampoco habría necesidad de coordinar [48].

Si existe la necesidad de coordinar también existe la necesidad de tomar decisiones que deben ser aceptadas, de forma directa o indirecta, por el grupo de actores que participan en el proceso. Por ejemplo, si los actores coordinan su esfuerzo hacia la solución de un problema deben estar de acuerdo en que ese problema es común a todos o conocer que acciones o actividades corresponde realizar a cada uno como resultado de su participación en el grupo. La coordinación requiere de procesos para la toma grupal de decisiones.

Si se retoma el ejemplo del rompecabezas colaborativo, los participantes están de acuerdo en que el objetivo colectivo es colaborar para resolver un rompecabezas; por lo tanto, los que tengan rol de jugador estarán a cargo de colocar piezas y los que tengan rol de profesor estarán a cargo de aprobar las solución propuestas. Como denota este ejemplo hay un acuerdo grupal previo, asumido de forma directa (por acuerdo) o indirecta (por imposición), sobre las actividades y el objetivo de la colaboración.

3.2.2. Canales de comunicación

La toma grupal de decisiones también tiene necesidades implícitas. Si hay necesidad de toma grupal de decisiones es inherente a este hecho una segunda necesidad: los miembros del grupo (o algunos por lo menos) requerirán canales de comunicación para el intercambio de información. Si los actores de un mismo grupo no tienen canales de comunicación para realizar los acuerdos mínimos necesarios que requiere la coordinación ésta última no puede existir. Si se piensa nuevamente en el rompecabezas groupware y en los acuerdos de la toma grupal de decisiones se tiene:

- El objetivo es resolver el rompecabezas.
- Los actores con rol de jugador estarán a cargo de colocar piezas.
- Los actores con rol de profesor estarán a cargo de revisar las soluciones propuestas.

Resulta obvio entender que antes de finalizar este proceso los diferentes actores establecieron además un proceso de comunicación. Sin este último proceso de por medio los actores no tendrían ningún acuerdo por lo que no existiría la coordinación y en consecuencia tampoco la colaboración. Por lo tanto se puede afirmar que la coordinación subyace sobre un proceso de comunicación que es vital para su existencia.

3.2.3. Protocolos comunes para intercambio de información

Tan vital para la coordinación es la comunicación como para la comunicación la existencia de un lenguaje común para el intercambio de información entre emisores y receptores. Problema clásico dentro de las comunicaciones independientemente del dominio de que se trate.

Comunicar no sólo representa intercambiar información entre dos o más puntos. Intercambiar información no basta si esta información no tiene la interpretación adecuada en todos los agentes involucrados en el proceso.

El intercambio de información en un proceso de coordinación puede tener como lenguaje común el habla o simplemente la habilidad de cada uno de los actores para percibir objetos que son comunes a todos. Si se extrapola esta situación a un escenario virtual (por ejemplo el rompecabezas groupware) esta percepción común de los objetos se da de forma abstracta. En este caso, el software de cada actor tiene código (interfaces, clases, procedimientos, etc.) que interpreta de forma común la información que se intercambia. Esto garantiza que la información en cada actor, emisor o receptor, se interpreta de forma común.

3.3. Dependencias inherentes del modelo

Dentro de un modelo de coordinación las directrices que guían o describen de que forma se debe reaccionar ante cada escenario distinto de colaboración se conocen como *política de coordinación*. En los sistemas groupware el conjunto de políticas establecidas representa el modelo global de coordinación del sistema. Y el modelo de coordinación de un sistema groupware permite controlar el flujo de datos a nivel de actividades o acciones.

Según [19] una política de coordinación es:

> El conocimiento sobre la forma de responder a ocurrencias particulares dentro de una situación de colaboración.

Las políticas poseen el conocimiento para reconocer una determinada ocurrencia y saben cómo manejarla. Los sistemas groupware se caracterizan por presentar cambios constantes que en algunos casos, si no son gestionados adecuadamente, puede desembocar en situaciones críticas. Por lo tanto, una política tiene como objetivo limitar selectivamente este dinamismo inherente a todos los sistemas groupware. Para llevar a cabo esta limitación selectiva, las políticas de coordinación, usualmente, median el acceso a los recursos o la ejecución de las propias acciones y actividades de un sistema groupware.

Sin embargo, realizar estas tareas puede resultar complicado. Para entender la complejidad del problema de coordinación se propone llevar a un sistema groupware el siguiente escenario de colaboración (de ocurrencia común en el mundo real):

> Necesito trabajar en mi tesis, solo, y en la escritura de un documento, junto con mi equipo de trabajo. Todo durante la próxima semana. Si durante este tiempo me vienen a buscar no estoy disponible, sólo que fuera mi familia o se tratara de un asunto urgente. Sin embargo, si es mi tutor y quiere revisar mis avances de tesis

> y, además, no estoy atrasado con la escritura del documento, estoy disponible. De lo contrario no recibiré a mi tutor, sin embargo, le daré acceso a mi oficina para que pueda consultar él mismo mis avances de tesis.

Coordinar un problema de cooperación grupal en el mundo real puede parecer trivial. Sin embargo, el ejemplo anterior muestra lo complicado que puede ser llevarlo a un sistema de cómputo. En el ejemplo encontramos que implementar una política que resuelva de forma eficiente el problema debe considerar actividades multiusuario y monousuario interdependientes, control de acceso a recursos, etc.

Esto quiere decir que, por un lado, el modelo de coordinación (conjunto de políticas) de un sistema groupware debe ser suficientemente *expresivo*, y por el otro, debe estar suficientemente *integrado* con los datos sobre el entorno de colaboración [19].

Si a este problema se suma que: las políticas deben ser *flexibles*, para poder adaptarse a los típicos cambios repentinos de cualquier escenario de colaboración, y, que el modelo de coordinación debería estar *desacoplado* de la lógica de las aplicaciones para simplificar su mantenimiento y reutilización [21], se tiene un problema cuya complejidad aumenta en varias dimensiones.

3.3.1. Expresivo

Un primer problema que se presenta cuando se requiere implementar políticas de coordinación en un sistema es: ¿Cómo se puede diferenciar a los actores para concederles distintos privilegios de acceso a los mismos recursos? Por ejemplo, si se piensa en un juego de rompecabezas donde las piezas se reparten en dos grupos de jugadores, donde cada uno de éstos es responsable de colocar exclusivamente las piezas que le corresponden se tiene que una política que no sea capaz de distinguir, al menos, entre dos tipos de actores distintos, los del grupo uno y los del grupo dos, no será útil.

Los sistemas colaborativos utilizan un principio de asignación de políticas a grupos de usuarios. Estos grupos se denominan *roles*. Un rol, entonces, representa una categoría o grupo de usuarios dentro de una población mayor donde todos utilizan la misma aplicación. En realidad el problema de distinguir entre distintos grupos de actores es un problema de expresividad. Muchas aplicaciones sólo proveen un control de acceso a recursos que puede ser otorgado o restringido de forma global [1]. En estas aplicaciones resulta imposible otorgar privilegios o denegar acciones a clases de actores. Sin embargo, como muestra el ejemplo del rompecabezas, el espectro de políticas que puede definirse resulta limitado para muchas aplicaciones.

[1] Ejemplos de estas aplicaciones son: [82][50] (tomado de [19]).

Soportar políticas de coordinación suficientemente expresivas es complicado y soportar roles de actores tampoco resulta suficiente para muchos escenarios de colaboración. Por ejemplo, si ahora se quiere implementar una política para coordinar el siguiente escenario de colaboración:

> No quiero que alguien que no sea parte de mi equipo de trabajo conozca el trabajo que estoy realizando.

Para este caso la definición de roles no es suficiente. La política requiere ahora la expresividad suficiente para conocer sobre que se está trabajando. Si el objeto sobre el que se trabaja cambia, la política debería aplicar el cambio también.

3.3.2. Flexible

Las políticas que coordinan la colaboración humana, al igual que ésta, tienen un carácter altamente dinámico. Por tal motivo son capaces de adaptarse a los cambios de escenarios que se les presentan. Proveer de esta flexibilidad a las políticas que coordinan los escenarios de colaboración en los sistemas groupware es fundamental. Una política que no presenta flexibilidad suficiente para adaptarse a los cambios que se presentan en su entorno puede llegar a limitar las interacciones y, por lo tanto, también la propia colaboración. Por ejemplo, supóngase que se quiere implementar una política de coordinación para el siguiente escenario de colaboración:

> Se tiene un juego de rompecabezas. Para colocar las piezas durante los primeros quince minutos el turno de los jugadores se asigna de forma secuencial. El resto del tiempo, el jugador que haya colocado más fichas en los quince minutos previos se convierte en el líder, a partir de ese momento, decide que jugador tiene el turno.

Para este caso un mismo jugador es representado dentro de la política por dos roles. Sin embargo, este problema va más allá de soportar actores con varios roles. El rol de líder se decide mientras se está armando el rompecabezas (en tiempo de ejecución de la aplicación) y no puede ser asignado de forma previa, por las propias restricciones del escenario de colaboración. Para resolver este problema las políticas de coordinación deberían soportar asignaciones o cambios de roles en tiempo de ejecución.

Algunos sistemas como [44][19] proporcionan políticas de coordinación que permiten definición dinámica de roles. Esto permite ofrecer escenarios de colaboración más reales que pueden ser coordinados por una o un conjunto de políticas de forma simultanea[2].

[2]Como muestra el ejemplo del rompecabezas, la mayor parte de los escenarios de colaboración reales son coordinados por un conjunto de políticas que pueden ser adecuadas, cambiadas

3.3.3. Integrado

El término *Políticas de coordinación integradas* hace referencia a la capacidad de las políticas para acceder al estado de su entorno. Cualquier tipo de actividad colaborativa tiene lugar en un entorno. Este entorno esta conformado por todo lo que rodea el escenario donde la colaboración tiene lugar: objetos, actores, actividades, la forma en la que interactúan unos con otros, la conciencia de grupo, o cualquier otro tipo de información que pueda afectar el estado del propio entorno.

Integrar las políticas de coordinación con esta información es necesario para la colaboración en diversas situaciones. Por ejemplo, supóngase que se quiere implementar una política de coordinación para el siguiente escenario de colaboración:

> Mientras estoy trabajando en mi tesis no estoy disponible. Sólo si alguien me busca más de tres veces en el mismo día porque probablemente se trate de un asunto urgente.

En este escenario la política debe tener acceso a los eventos que han sucedido en el pasado. Esa información debe formar parte del entorno. Por lo tanto el entorno deberá ser capaz de actualizar y reflejar su estado dinámicamente. Así, la política será responsable sólo de consultar esta información y responder en función de ello. En resumen, la política deberá conocer el estado del entorno en cada momento para determinar cuando una misma persona ha realizado la actividad "buscar al sujeto"por tercera vez.

3.3.4. Desacoplado

Otro de los problemas de los sistemas groupware es cómo proporcionar políticas de coordinación "holgadamente acopladas"[3]. [45] resalta este problema afirmando que la mayor parte de los sistemas CSCW tradicionales incrustan el código de coordinación junto al de la aplicación. Esto representa un problema serio al tiempo de querer extender la lógica, sea la de la propia aplicación o la de de los servicios de coordinación, porque existe una dependencia íntima entre ambas.

Proporcionar políticas de coordinación desacopladas de las aplicaciones colaborativas implica tener software que puede adaptarse con facilidad a otras aplicaciones. Las políticas deberían ser independientes de la aplicación que coordinan. Desacoplar la lógica de coordinación de las aplicaciones es un punto clave para proporcionar políticas de coordinación adaptables y reutilizables.

---

o extendidas mientras la colaboración se está dando.

[3]Del término en inglés *Loosely coupled*.

Relacionados con la solución de este problema se pueden destacar diversos esfuerzos. En [21], por ejemplo, se propone que desde la propia concepción de los sistemas groupware el diseño esté en función de tres submodelos principales: el ontológico, el modelo de coordinación y el modelo de interfaces. En [44], en cambio, se proporciona un lenguaje para especificar políticas de coordinación independientes de las aplicaciones. En [67] se propone un modelo de diseño para sistemas groupware que desacopla la coordinación de las aplicaciones utilizando una técnica de programación conocida como AOP (Programación Orientada a Aspectos)[4] [63]. Esta última propuesta es coincidente con la solución propuesta por la presente tesis a partir del hecho que ambas utilizan AOP para incorporar modelos de coordinación en las aplicaciones groupware.

3.4. Conciencia del espacio de trabajo

3.4.1. Necesidad de integrar la coordinación de acciones y la conciencia del espacio de trabajo

Anteriormente se ha definido el término coordinación como la acción de gestionar las dependencias que surgen entre diversas actividades [48]. Donde tales dependencias pueden quedar de manifiesto a dos niveles:

- A nivel de actividad.
- A nivel de acción.

El caso particular de la gestión de dependencias entre acciones, tema que atañe a la realización de este trabajo, pone en evidencia un requisito fundamental para lograr la coordinación: conocer o tener conciencia de las acciones que se producen (conciencia de espacio de trabajo).

> La conciencia del espacio de trabajo es el entendimiento de las interacciones (acciones colaborativas) de otra persona con el espacio compartido. A un nivel simple esto involucra el conocimiento de quienes están presentes, donde están ellos trabajando y lo que están haciendo [31].

La conciencia del espacio de trabajo y la coordinación son procesos que deben coexistir para que una colaboración efectiva pueda darse. En el caso particular de la coordinación la conciencia del espacio de trabajo es un proceso prioritario y siempre requerido ya que asiste la gestión de dependencias entre acciones y actividades.

[4] Del término en inglés *Aspect Oriented Programming*.

> La información de conciencia de grupo es siempre requerida para realizar la coordinación de actividades de grupo (cualquiera que sea el dominio de aplicación) [16].

> La información de conciencia de espacio de trabajo es útil para diversas actividades de la colaboración entre ellas la coordinación de acciones [31].

> La conciencia del espacio de trabajo asiste la coordinación de tareas y recursos [30].

Un proceso de colaboración para ser efectivo requiere de la existencia de procesos de coordinación. Un proceso de coordinación, como se ha detallado en esta sección, requerirá de igual forma de otro proceso que provea información relacionada con la conciencia de grupo.

En el caso de la coordinación de acciones (coordinación a nivel de objeto) la información de conciencia de grupo que se requiere prioritariamente es la información relacionada con las acciones que acontecen (conciencia del espacio de trabajo).

3.4.2. Marco conceptual para la conciencia del espacio de trabajo

La conciencia y el trabajo en grupo son temas largamente estudiados dentro de las disciplinas científicas relacionadas con la colaboración y particularmente las relacionadas con el desarrollo de sistemas groupware. Desarrollar mecanismos que soporten la conciencia de grupo en sistemas groupware es una tarea que impone dos dificultades serias a los desarrolladores [30]:

1. ¿Qué información debería el sistema groupware capturar sobre las interacciones que acontecen en el espacio de trabajo?
2. ¿Cómo debe presentarse esta información a los actores?

Atendiendo a esta necesidad se han desarrollado herramientas (marcos conceptuales) que permiten la clasificación y el análisis de la información relacionada con la conciencia de grupo.

Entre los trabajos más destacados se tiene [31]. En éste se propone un marco conceptual (ver cuadro 3.2) que identifica los diversos elementos relacionados con el concepto de conciencia del espacio de trabajo. Los elementos son clasificados en categorías, de acuerdo al tipo de información que proveen, y, además, son asociados de forma individual con una pregunta que describe la información que cada elemento representa dentro de la conciencia del espacio de trabajo.

CATEGORÍA	ELEMENTO	PREGUNTA
Quién	Presencia	¿Hay alguien en el espacio de trabajo?
*	Identidad	¿Quienes están participando?
*	Autoría	¿Quién está haciendo esto?
Qué	Acción	¿Qué están haciendo?
*	Intención	¿De qué objetivo es esa acción?
*	Objeto	¿Sobre qué objeto están trabajando?
Dónde	Localización	¿Dónde están trabajando?
*	Mirada	¿Hacia dónde están viendo?
*	Vista	¿Qué pueden alcanzar a ver?
*	Alcance	¿Hasta donde pueden llegar?
Cuándo	Historial de eventos	¿Cuándo ocurrió ese evento?
Cómo	Historial de acciones	¿Cómo sucedió esa operación?
*	Historial de objetos	¿Cómo ese objeto pasó a ese estado?

Cuadro 3.2: Marco conceptual de la conciencia del espacio de trabajo (actividades síncronas).

Esta clasificación es una de las más aceptadas y como consecuencia actualmente es coincidente en diversos trabajos de investigación relacionados con el estudio de la conciencia del espacio de trabajo como [16][54][82][32]. [31] afirma, por ejemplo, que para cualquier tipo de trabajo colaborativo ésta (la información sintetizada dentro del marco) es la información de interés para todos los diferentes actores, y, por lo tanto, ésta es la información que los diseñadores de sistemas groupware siempre deberían considerar.

3.5. Acciones/Interacciones

3.5.1. Interactividad en los sistemas groupware

Se ha definido el concepto de interacción como un tipo particular de acción que siempre conlleva una reacción o consecuencia. Sin embargo en el caso de los sistemas groupware este concepto va más allá de la simple explicación lingüística citada anteriormente.

El estudio de las interacciones dentro de los dominios relacionados con la colaboración es un tema que ha sido largamente investigado. Tareas como proveer de una definición al término o identificar el umbral que existe entre el paso de acciones a interacciones (punto clave para la definición) son hechos hasta ahora inconclusos.

En el caso de los sistemas groupware el paso de una acción a una interacción es atribuido a la generación de participación de diferentes actores (interactividad). La dificultad de este hecho recae en evaluar si el grado de interactividad que ha generado una acción es suficiente para considerarla una interacción o no. Pero la evaluación de la interactividad en un contexto colaborativo virtual es complicado.

Contra este problema existen soluciones que proponen determinar el grado de interactividad a partir de parámetros cuantitativos. Por ejemplo, cuantas reacciones o participaciones se han generado. Sin embargo, también existen otras soluciones que proponen determinar la interactividad a través de parámetros cualitativos. Por ejemplo, la influencia de la acción en los procesos cognitivos de los diferentes actores. Para este caso, el hecho de no tener reacción podría denotar mayor interactividad que el hecho de tenerla.

En [53][51] se propone un método para apoyar el análisis de las interacciones en aplicaciones CSCL. Parte de la problemática central consiste en proveer un modelo computacional a través del cual representar las interacciones que se suscitan en un escenario de colaboración. Como parte de ese trabajo, además, se propone una definición del término interacción que corresponde a la problemática antes mencionada.

Dado que el ámbito de estudio del presente trabajo corresponde al dominio de los sistemas CSCW/CSCL, y que la definición y representación computacional de interacción que se hace en [51] es extrapolable a los sistemas groupware en general, se propone adoptar ambos conceptos. Éstos mismos se ilustran a continuación.

Concepto de interacción

Según [51] una interacción es:

> Una interacción es una acción que afecta, o puede afectar, al proceso de colaboración. El requisito fundamental será que la acción en sí o su efecto pueda ser percibida por al menos un miembro del grupo o comunidad distinto del emisor [51].

A partir de esta definición, se puede considerar una interacción acción o consecuencia de ésta si:

1. Puede ser percibida por los demás participantes.
2. Se puede observar su efecto sobre algún objeto compartido.

Esta definición engloba diferentes tipos de interacción:

- Interacción o acción colaborativa directa. Este tipo de acción se caracteriza por tener un emisor, un receptor y un contenido.
- Interacción o acción colaborativa indirecta. Este tipo de acción se caracteriza por estar mediada por un objeto, y se produce a través de los denominados espacios de trabajo compartido.
- Participación. Representa una intervención genérica en el espacio de trabajo. De este tipo son las acciones que no tienen un destino conocido, aunque también pueden incluirse acciones directas e indirectas.

Dentro de los sistemas groupware las interacciones que se generan a través de las computadoras entre los diferentes actores del espacio de trabajo compartido pertenecen al grupo de las interacciones indirectas. Se llaman interacciones indirectas porque los actores, en realidad, utilizan objetos compartidos para generar interactividad con otros actores.

Modelo de representación de acciones

En [51] se propone la representación de acciones a través de los elementos:

- Acción. Acción que se realiza.
- Objeto. Objeto sobre el que se realiza la acción.
- Actor. Actor que realiza la acción.

Por lo tanto, el modelo de representación de una acción queda dado por la relación:

```
Accion(Objeto, Actor)
```

Este modelo propone anexar como información adicional los atributos particulares de cada objeto. Por ejemplo, en el caso de que un atributo del objeto fuera su posición el modelo quedaría de la siguiente forma:

```
Accion(Objeto, pos(X,Y), Actor)
```

El modelo genérico para la representación de una acción quedaría de la siguiente forma:

```
Accion(Objeto, Actor, Atr1, Atr2, ... AtrN)
```

3.5.2. Necesidad de generar eventos con niveles de abstracción más altos

Ofrecer mejores mecanismos de coordinación, y por consiguiente de conciencia de grupo, es un problema fuertemente ligado al tratamiento de las interacciones dentro de los sistemas groupware. La necesidad de ofrecer más y mejor información relacionada con las interacciones es una de las claves de este problema.

Dentro de un espacio de trabajo virtual las interacciones que se producen entre los diferentes actores tienen como origen el acontecimiento de un evento. Inicialmente este es un evento físico, por ejemplo la acción de un botón en un dispositivo de entrada como el teclado o el ratón. Estos dispositivos a su vez capturan el evento y lo convierten ahora en una interrupción de los procesos que se están ejecutando en la computadora.

En este caso, por tratarse de un proceso que pertenece a la ejecución de un sistema groupware, la interrupción se asocia además a la interfaz del sistema. Esta asociación permite obtener un *evento de interfaz*. Un evento de interfaz provee información no sólo relacionada con la tecla o el botón del ratón que se presionó sino también con el elemento de la interfaz que supuestamente se utilizó (un botón, un menú, etc.), el número de pulsaciones que se hicieron a este elemento en un intervalo de tiempo dado, etc.

Por ejemplo, el evento pulsar el botón izquierdo del ratón sobre un menú provee información sobre el número de pulsaciones y la ubicación en la pantalla donde se hicieron tales pulsaciones, además de la información relacionada con el dispositivo (en este caso el ratón). Este evento de interfaz es notablemente más complejo y se compone de varios eventos físicos[5].

La nueva composición o abstracción del evento le permite ofrecer más información al proceso en ejecución. En realidad los sistemas deberán tener procedimientos específicos para capturar estos eventos. Cuando un procedimiento captura un evento obtiene de éste toda la información posible. Esta información, o mejor dicho la semántica de esta información, es lo que permite al proceso interpretar los cambios de estado que deben originarse a partir de la ocurrencia del evento.

En el caso de los sistemas groupware la información que proveen los eventos de interfaz debe ser interpretada en cambios de estado del espacio de trabajo compartido. Sin embargo, el problema es que estos sistemas trabajan a un nivel de abstracción mucho más alto. Sus procesos están orientados a la ejecución de acciones, interacciones, actividades, etc. y la traducción de éstas en eventos de interfaz provee información muy pobre que normalmente no satisface los requerimientos mínimos para proveer mecanismos relacionados con la conciencia de grupo o la coordinación.

Una de las diferencias más notables entre colaborar a través de espacios físicos y a través de espacios virtuales es precisamente la riqueza de las interacciones que se tiene. Esto deriva en la urgente necesidad de proveer a los sistemas groupware de eventos semánticamente más ricos, cuyas abstracciones provean información suficiente sobre las acciones y actividades que acontecen.

En [51] se hace una propuesta (Cuadro 3.3) de los diferentes niveles de abstracción en que que se pueden presentar las interacciones cuando se realizan a través de una computadora. Esta propuesta permite ilustrar el problema de abstracción de eventos que se tiene dentro de los sistemas CSCW/CSCL. Mientras los niveles de abstracción que se requieren en este dominio pertenecen a la capa más alta: las interacciones (incluso por arriba de las acciones); la mayor parte de los sistemas groupware sólo ofrecen abstracciones de eventos a un nivel de interfaz (muy por debajo de lo que se requiere).

[5]Para originarlo debe pulsarse el botón primero hacia abajo y después hacia arriba en el mismo lugar de la pantalla y en un intervalo de tiempo t.

ABSTRACCIÓN	EJEMPLO
Interacción	Conflicto, acuerdo, participación, etc.
Acción	Se mueve pieza del rompecabezas
Evento de interfaz	Se presiona un boton
Evento físico	Dedo presiona tecla

Cuadro 3.3: Niveles de abstracción en los que se puede presentar una interacción.

3.6. Discusión

A lo largo de este capítulo se ha expuesto un marco conceptual de coordinación y conciencia de grupo en sistemas CSCW/CSCL. Este marco destaca como necesidades primarias:

Desacoplar modelos de coordinación y conciencia de grupo

Desacoplar el modelo de coordinación y conciencia de grupo de las aplicaciones CSCW/CSCL permite mejorar notablemente el grado de reutilización del software. Dotar de servicios de coordinación y conciencia de grupo a los sistemas CSCW/CSCL tiene un costo de desarrollo y mantenimiento alto. Desacoplar la lógica de los servicios de colaboración de la lógica de las aplicaciones no sólo simplifica notablemente su mantenimiento sino que contribuye fuertemente a su posterior reutilización.

Proveer modelos de coordinación y conciencia de grupo extensibles

En la actualidad, dado su alto costo de desarrollo y mantenimiento, los sistemas CSCW/CSCL conllevan implícitamente la necesidad de convertirse en sistemas de fácil extensión. Dado que en el momento de su diseño y desarrollo este tipo de sistemas no pueden contemplar todos los escenarios futuros en los que serán requeridos, es fundamental integrar mecanismos a través de los cuales sea posible la extensión futura de sus capacidades.

Proveer modelos de coordinación y conciencia de grupo flexibles

El carácter altamente dinámico que presentan los sistemas CSCW/CSCL exige modelos de coordinación y conciencia de grupo flexibles, que sean capaces de adaptarse a los cambios que se presentan en su entorno.

Utilizar modelos de coordinación y/o conciencia de grupo demasiado rígidos limita fuertemente la interacción entre actores; y este hecho suele ser perjudicial para la colaboración.

Integrar modelos de coordinación y conciencia de grupo

La coordinación y la conciencia de grupo son factores de mutua influencia dentro de la colaboración. Mientras la coordinación (a nivel de acciones) se encarga, precisamente, de manejar todas las dependencias que surgen de las acciones generadas, la conciencia de grupo constituye el conocimiento previo y necesario sobre la existencia de tales acciones.

Proveer eventos a un nivel de abstracción más alto

La mayor parte de los sistemas CSCW/CSCL que ofrecen servicios de coordinación o conciencia de grupo utilizan abstracciones de eventos a un nivel de interfaz de usuario. Desafortunadamente estas abstracción representan un *déficit* de información porque no permiten, o muy difícilmente lo hacen, obtener la información necesaria (tipo de acción, autoría, objeto, lugar, tiempo, actividad, etc.) para proveer mecanismos eficientes de conciencia del espacio de trabajo y coordinación de acciones.

El siguiente capítulo recoge todas las necesidades expuestas anteriormente y las aplica al diseño e implementación de un marco CSCW/CSCL: AORTA. AORTA en un marco basado en componentes de software que ofrece servicios, de coordinación y conciencia de grupo, a través de los cuales es posible simplificar el ciclo de desarrollo y mantenimiento de las aplicaciones CSCW/CSCL.

Capítulo 4

AORTA: Arquitectura desacoplada y orientada a acciones para soportar coordinación y conciencia de grupo

Dentro del dominio de los sistemas colaborativos se han desarrollado diversas soluciones de reutilización de software basadas en componentes. Particularmente, algunos marcos para el desarrollo de aplicaciones CSCW/CSCL [26][28] proveen servicios de colaboración, basados en componentes, que son reutilizados exitosamente dentro de múltiples aplicaciones.

El manejo de grupos, el manejo de sesiones, el manejo de objetos compartidos, entre otros, son claros ejemplos de algunos de los requerimientos que los sistemas colaborativos resuelven con soluciones más y mejor reutilizables cada vez. Incluso, podría afirmarse que aquellos requerimientos que son considerados primarios dentro del soporte de un espacio de trabajo compartido encuentran soluciones de reutilización cada vez más satisfactorias dentro de los marcos.

Desafortunadamente, también existen otros requerimientos, no menos importantes, como son la coordinación y la conciencia de grupo que han recibido poca atención por parte de los marcos.

Si bien es cierto que estos servicios no resultan fundamentales para soportar un espacio de trabajo compartido sí lo son para que el proceso de colaboración se desarrolle de forma adecuada.

> Responder de qué forma las interdependencias que surgen entre actividades pueden ser encausadas a la solución de una meta común es una cuestión tan vital como compleja dentro del proceso de colaboración. Precisamente a esta gestión de dependencias de actividades es lo que se le conoce como **coordinación** [48].

> La **conciencia de grupo** juega un papel fundamental dentro de la colaboración. Permite la coordinación de tareas y recursos, además, constituye el puente de enlace entre actividades individuales y compartidas [16].

Con el objetivo de atender a esta importante necesidad de incorporar en los marcos servicios que ofrezcan el soporte adecuado de coordinación y conciencia de grupo se presenta AORTA *(**A**ction-oriented dec**O**upled **A**R**c**hitecture for coordina**T**ion and **A**wareness)* [61]. AORTA es un marco CSCW/CSCL que provee servicios de coordinación a nivel de objetos y de conciencia del espacio de trabajo a marcos y/o aplicaciones CSCW/CSCL.

4.1. Características funcionales

AORTA es un marco CSCW/CSCL. Sin embargo, más allá de ser un marco, y debido a sus características funcionales (ver cuadro 4.1), AORTA puede considerarse un complemento funcional para marcos CSCW/CSCL existentes.

La idea fundamental detrás del uso de AORTA es la reutilización: **no desarrollar para cada aplicación colaborativa un soporte distinto de coordinación y conciencia de grupo.** En vez de esto los desarrolladores de aplicaciones CSCW/CSCL pueden reutilizar dentro de cada aplicación el soporte de coordinación y conciencia de grupo que AORTA ofrece. Hecho que les permite, además de ahorrar una gran cantidad de trabajo, enfocar toda su atención en el desarrollo y mantenimiento de funcionalidades propias de sus aplicaciones (ver figura 4.1).

4.1.1. Arquitectura replicada

La arquitectura replicada de AORTA está pensada en ofrecer a aplicaciones colaborativas síncronas una herramienta ágil para la toma de decisiones. Cada decisión sobre coordinación y conciencia de grupo se resuelve de forma local y con independencia de la red para garantizar el tiempo de respuesta y para no obstruir la colaboración síncrona. (utilizando información previamente recibida desde las otras aplicaciones).

ASPECTO	DETALLES
Orientación a acciones	Uso de abstracciones de eventos a nivel de acciones. La acción constituye la unidad básica para el intercambio de información entre las aplicaciones y AORTA
Soporte integrado de coordinación y conciencia de grupo	Las decisiones de coordinación en AORTA se nutren de la información que se genera sobre el espacio de trabajo compartido
Servicios basados en componentes	La extensión o adaptación funcional de AORTA se da, sin necesidad de cambio alguno, a través de la inclusión de componentes de software (llamados políticas)
Uso de patrones de diseño	La utilización de patrones de diseño de software permiten desacoplar la lógica de coordinación y conciencia de grupo que provee AORTA de la lógica de las propias aplicaciones

Cuadro 4.1: Aspectos funcionales más destacados dentro de la arquitectura de AORTA.

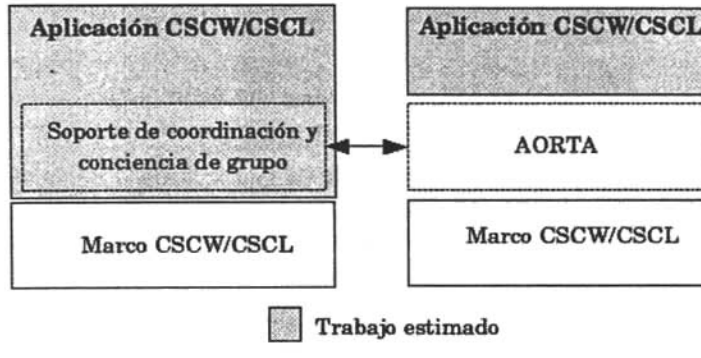


Figura 4.1: Idea fundamental en el uso de AORTA.

El trabajo que un desarrollador requiere para incorporar servicios de coordinación y conciencia de grupo en cualquier aplicación es costoso. La idea fundamental detrás del diseño de AORTA es proveer a los desarrolladores una herramienta de reutilización eficaz que les permita centrar su trabajo en aquellos requerimientos que son propios de sus aplicaciones.

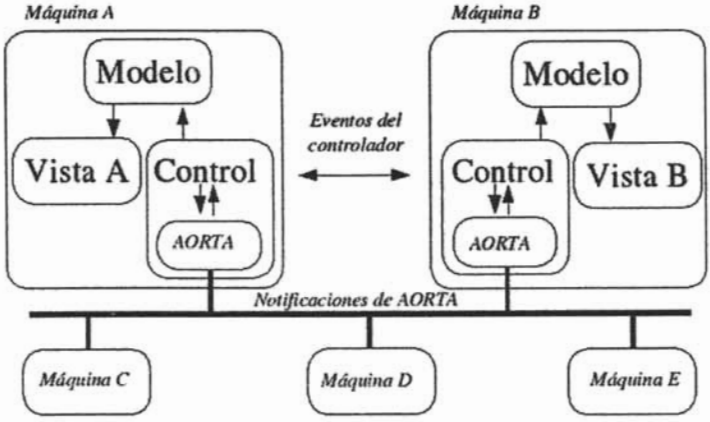


Figura 4.2: AORTA dentro de una aplicación CSCW/CSCL con arquitectura MVC replicada.

Esta figura muestra una aplicación MVC replicada que está integrada con AORTA. Como detalla esta figura las llamadas y modificaciones al modelo de la aplicación, a diferencia de cualquier otra aplicación MVC convencional, son realizadas por AORTA y no por la propia aplicación (AORTA está integrada como parte del control de la aplicación).

AORTA está diseñada para proveer sus servicios, principalmente, a aplicaciones colaborativas síncronas que siguen las variantes híbrida o replicada del patrón arquitectónico MVC [88]. La integración entre AORTA y las aplicaciones se da a través del controlador de estas últimas (ver figura 4.2).

En una arquitectura MVC el controlador es el elemento responsable de manejar las posibles modificaciones al modelo de una aplicación. Sin embargo, el uso de AORTA modifica parcialmente este comportamiento. A diferencia de una aplicación MVC convencional, una aplicación MVC que utiliza AORTA delega a ésta última el manejo (total o parcial) de posibles modificaciones en su modelo.

Para lograr esto, cada una de las replicas de AORTAS interactúa sobre cada una de las replicas del controlador de la aplicación. Hecho que obliga a las aplicaciones, por compatibilidad con AORTA, a replicar el controlador de forma total o parcial. Tener replicas del controlador de una aplicación garantiza a AORTA que el tiempo de respuesta en la toma de decisiones no obstaculizará el flujo natural de la colaboración.

Tomando en cuenta que AORTA trabaja exclusivamente sobre aplicaciones colaborativas síncronas el tiempo de respuesta constituye un punto crítico en su funcionamiento. Es por ello que aplicaciones con otras variantes del patrón MVC (híbridas o distribuidas) no resultan adecuadas para trabajar con AORTA.

4.1.2. Orientación a acciones

Una de las diferencias más importantes que existe entre una colaboración física (donde los participantes están físicamente presentes en el espacio de trabajo compartido) y una colaboración virtual (donde los participantes colaboran a través del una aplicación CSCW/CSCL, por ejemplo) es el número de interacciones que se producen en cada una. En consecuencia, a diferencia de la colaboración física, la colaboración virtual produce interacciones con una amplitud semántica considerablemente menor.

Las aplicaciones CSCW/CSCL, en general, proveen información sobre las interacciones que acontecen a través de la emisión de eventos de bajo nivel (por ejemplo eventos de interfaz). Sin embargo, el problema es que estos eventos pertenecen a un nivel de abstracción más bajo que los procesos de las aplicaciones implicadas (procesos normalmente orientados a la ejecución de acciones, interacciones, actividades, etc.), y esto en muchos casos constituye un serio déficit de información.

La orientación a acciones, sin duda, es uno de los conceptos claves detrás de la arquitectura de AORTA porque precisamente hace referencia a esta necesidad de las aplicaciones CSCW/CSCL de proveer interacciones a un nivel de abstracción más alto. AORTA utiliza, a diferencia de otras propuestas, una aproximación de abstracciones de eventos a nivel de acciones. Por lo tanto, para AORTA, la acción constituye la unidad básica para el intercambio de información con las aplicaciones.

AORTA posee una arquitectura orientada a acciones. Esto significa que AORTA coordina (servicios de coordinación) y hace conscientes a sus participantes (servicios de conciencia de grupo) de las acciones que acontecen en el espacio de trabajo. Pero, ¿qué son estas acciones?

Antes de que se defina el término acción es importante destacar que se trata de un concepto confuso y que desde luego no se cuenta con una sola definición en la literatura relacionada. Por tal motivo, y para los fines de este trabajo, se han decidido adoptar algunas de las definiciones que aparecen en trabajos de investigación relacionados con el estudio de las interacciones colaborativas [52][57].

De acuerdo a estas definiciones, lo que AORTA en realidad coordina y sobre lo que hace conscientes a los participantes son: interacciones colaborativas indirectas. Donde:

> Una interacción colaborativa es una acción que afecta (o puede afectar) el proceso de colaboración [57].

Cuando esta interacción se realiza con la ayuda de un artefacto tecnológico, por ejemplo una computadora, se le llama interacción colaborativa indirecta.

Y una interacción colaborativa indirecta (y en consecuencia la acción que la origina) se caracteriza por un rol (dado por el participante que genera la acción), un objeto compartido (el objeto sobre el que la acción se realiza), una operación lo que se realiza sobre el objeto) y el tiempo (el instante preciso en el que el usuario realiza la operación sobre el objeto) [57].

En este contexto una acción debe ser entendida como el evento observable de una aplicación que se expresa en términos significativos para esa misma aplicación. Por ejemplo, un evento (un cambio en el texto que genera un editor colaborativo) puede constituir la acción *modificar documento* o *generar documento*. Acción què se convierte en una interacción desde el momento en que potencialmente puede afectar a otro usuario.

El uso de AORTA tiene algunas implicaciones en el proceso de desarrollo de una aplicación CSCW/CSCL. A causa del concepto de acción, o mejor dicho de sus implicaciones de diseño dentro de la arquitectura de AORTA, los desarrolladores están obligados a traducir los eventos que se generan dentro de sus aplicaciones en acciones (eventos observables y significativos para las aplicaciones).

Por ejemplo: una aplicación colaborativa para la solución de rompecabezas. En este caso el desarrollador es responsable de traducir eventos de la aplicación que no tienen ningún significado para ésta (por ejemplo un evento de ratón) en las acciones que son importantes para la aplicación: por ejemplo *seleccionar pieza* o *soltar pieza*.

Aún cuando el uso de acciones parece ser un diferencia mínima no lo es. Las acciones son eventos más cercanos al nivel de abstracción de los desarrollador de aplicaciones CSCW/CSCL. Para un desarrollador es mucho más fácil diseñar aplicaciones pensando en las acciones de la aplicación, y en la influencia de éstas dentro de la colaboración, que pensar en eventos de bajo nivel de abstracción (como eventos de ratón, teclado o interfaz).

4.1.3. Soporte integrado de coordinación y conciencia de grupo

AORTA es un marco que ofrece servicios *integrados* de coordinación a nivel de objetos y conciencia del espacio de trabajo a aplicaciones CSCW/CSCL.

Dentro de la definición anterior es muy importante resaltar el término *integrado* porque hace referencia al hecho d« que las decisiones de coordinación se nutren de la información que se genera sobre el espacio de trabajo y viceversa.

Diversos e importantes autores destacan la mutua dependencia que existe entre la coordinación y la conciencia de grupo y la importancia que ésta tiene para el proceso de colaboración:

> La información de conciencia de grupo es siempre requerida para realizar la coordinación de actividades de grupo (cualquiera que sea el dominio de aplicación) [16].

> La información de conciencia de espacio de trabajo es útil para diversas actividades de la colaboración entre ellas la coordinación de acciones [31].

Para entender la importancia de esta relación dentro del proceso de colaboración se propone el siguiente ejemplo:

Se tiene un editor colaborativo síncrono. Para definir que usuario tiene derechos de escritura sobre el texto en cada momento el editor utiliza una política de coordinación de turnos. Esto significa que si un usuario edita el texto no podrá tener permisos sobre éste hasta que el resto de usuarios participantes hayan colaborado en el proceso de escritura. El editor es capaz de notificar los intentos de un usuario para realizar una modificación ilegal o fuera de turno (genera información de conciencia de grupo). Además, su servicio de coordinación es capaz de acceder, en todo momento, a esta información sobre el espacio de trabajo (integra servicios de coordinación y conciencia de grupo). Gracias a esta integración la política de coordinación es capaz de modificar su estado dinámicamente. Por ejemplo, la política podría omitir el turno de cualquier usuario que haga x intentos por modificar el texto fuera de turno.

Como este escenario existen infinidad de ejemplos en los que la integración entre conciencia de grupo y coordinación no sólo es útil sino fundamental para asistir de forma correcta el proceso de colaboración.

Atendiendo a esta necesidad AORTA integra sus servicios de coordinación y conciencia de grupo. Integración que le permite, por un lado, proveer políticas de conciencia de grupo capaces de notificar de forma síncrona las decisiones de coordinación que se toman, y por el otro, políticas de coordinación capaces de modificar su estado en función de la información que AORTA genera sobre el espacio de trabajo.

CARACTERÍSTICA	DESCRIPCIÓN
Reutilización	AORTA permite la reutilización de la misma política en diversas aplicaciones CSCW/CSCL
Adaptación	Las políticas pueden ser cambiadas durante el tiempo de ejecución de una aplicación para responder a cambios de estado del escenario de colaboración
Extensión	El funcionamiento de AORTA puede ser extendido, sin cambios en su arquitectura, a través de la inclusión de nuevas políticas

Cuadro 4.2: Importancia de las políticas en la arquitectura de AORTA.

4.1.4. Servicios basados en componentes

Los servicios de coordinación a nivel de objetos y conciencia del espacio de trabajo que AORTA provee están basados en el uso de políticas. Las políticas son un conjunto de reglas (de coordinación o conciencia de grupo) encapsuladas dentro de un componente de software. Dentro de la arquitectura de AORTA estos componentes o políticas constituyen herramientas poderosas de reutilización, adaptación y extensión (ver cuadro 4.2).

Las políticas son ortogonales a las aplicaciones que las utiliza. Esto significa que AORTA permite el uso de una misma política dentro de más de una aplicación. Utilizar una misma política en varias aplicaciones es una tarea simple desde el punto de vista funcional. Sin embargo es importante mencionar que el potencial real de reutilización de cada política depende de su propio diseño (los programadores pueden desarrollar sus propias políticas).

Otro aspecto importante de AORTA es la adaptabilidad. Gracias a que las políticas pueden ser cambiadas durante el tiempo de ejecución de una aplicación AORTA puede ofrecer servicios de coordinación y conciencia de grupo capaces de adaptarse a escenarios de colaboración altamente dinámicos y complejos. Si hay un cambio de estado en la colaboración AORTA puede adaptarse de forma inmediata a este cambio mediante uno o varios cambios de política.

Las políticas son almacenadas en un espacio físico llamado repositorio. AORTA define dos repositorios: uno para las políticas de coordinación y uno más para las políticas de conciencia de grupo. Los repositorios son el medio a través del cual AORTA conoce *a priori* la ubicación exacta de las políticas. Los repositorios representan los puntos de extensión de AORTA. AORTA puede ser extendida a través del desarrollo e inclusión de nuevas políticas en los repositorios.

4.1.5. Servicios desacoplados

Los sistemas CSCW/CSCL tradicionales incrustan el código de los servicios de colaboración junto al de la lógica de las aplicaciones [45].

Lo anterior representa un problema grave ya que dificulta notablemente el mantenimiento no sólo de los servicios de colaboración sino de la propia lógica de las aplicaciones. Proveer servicios de colaboración desacoplados es fundamental para el ciclo de vida de las aplicaciones. Separar la lógica de los servicios de la lógica de las aplicaciones es una tarea tan compleja como necesaria para las aplicaciones colaborativas.

Atendiendo a este hecho AORTA ofrece una solución, basada en patrones de diseño, que permite desacoplar la lógica de coordinación y conciencia de grupo de la lógica de las aplicaciones.

Concretamente, esta solución consiste en proveer a las aplicaciones con claros puntos de corte en en el flujo normal de su ejecución. Posteriormente, estos puntos son aprovechados por AORTA para la incorporación de aspectos. Estos aspectos son el medio a través del cual AORTA introduce la lógica de servicios en las aplicaciones. Esta técnica de programación se conoce como: AOP (Programación Orientada a Aspectos) [63].

La utilización de ésta técnica conlleva a los programadores a considerar modificaciones de diseño que pueden resultar atípicas para ellos. Los programadores son responsables de identificar la ubicación de los puntos de corte dentro de sus aplicaciones.

Estos puntos de corte están determinados fundamentalmente por las acciones más importantes que tiene cada aplicación, o sea, aquellas acciones a las que el programador está interesado en incorporar aspectos de coordinación y conciencia de grupo.

Una vez identificados los puntos de corte los programadores deberán incorporar una llamada a AORTA justo antes de la ejecución de las acciones. En realidad, y como se detalla más adelante, la ejecución de las acciones es delegada a AORTA a través de esa llamada.

4.2. Arquitecura propuesta

AORTA presenta un arquitectura en capas que le permite limitar selectivamente la comunicación y el intercambio de información entre sus distintos elementos. Cada capa representa un módulo[1] de software con funciones específicas y completamente distintas a las de las capas restantes.

Por tal hecho, los elementos o componentes fuertemente acoplados pertenecen necesariamente a las mismas capas mientras que los elementos de distintas capas presenta un acoplamiento mínimo que se limita a elementos de capas contiguas (ver figura 4.3).

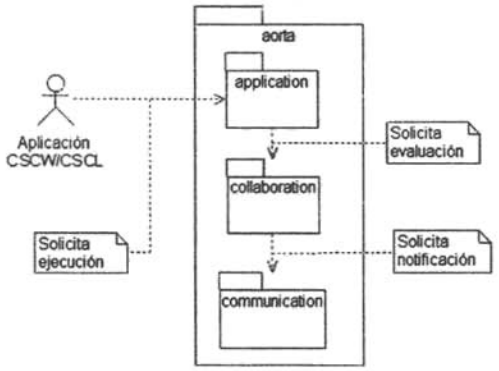


Figura 4.3: Dependencias funcionales de una aplicación con AORTA.

Esta figura describe las dependencias funcionales entre una aplicación CSCW/CSCL y cada una de las capas AORTA. La capa más alta, la capa de aplicación, es responsable de proveer de servicios a las aplicaciones. Sin embargo, para realizar este trabajo depende funcionalmente de su capa subyacente, la capa de colaboración (que contiene la lógica de los servicios de coordinación y conciencia de grupo), que a su vez depende de su capa subyacente o capa de comunicación. Como muestra la figura la arquitectura en capas le permite a AORTA aislar el acoplamiento can las aplicaciones dentro de la capa más alta.

Como cualquier arquitectura en capas AORTA pretende segmentarse en bloques con distintos niveles de abstracción. La idea principal es que la capa más alta, la capa de aplicación, que es la capa mediante la cual el marco se comunica con el exterior, se vea beneficiada ocultando toda la complejidad de interacción dentro de las capas inferiores (ver cuadro 4.3).

[1] El término módulo hace referencia a un conjunto de dos o más componentes fuertemente acoplados.

VENTAJAS	DESVENTAJAS
Mayor facilidad de reutilización de cada capa	Si no se logra la contención de un cambio se requiere un cambio en cascada
Mayor facilidad en la estandarización de las capas	Una arquitectura en capas puede significar pérdida de eficiencia
Las dependencias se limitan a capas contiguas	Se realiza trabajo innecesario o redundante en las capas intermedias
Los cambios se limitan a una o pocas capas	Es difícil diseñar correctamente la granularidad de las capas

Cuadro 4.3: Ventajas y desventajas de una arquitectura en capas.

Funcionalmente AORTA se compone por cuatro elementos principales (ver figura 4.4) que se distribuyen a lo largo de tres capas (ver figura 4.5):

1. Capa de aplicación.
   - Elemento AEE (*Action Execution Engine*). Punto de interacción entre AORTA y las aplicaciones. Las aplicaciones ejecutan sus propias acciones a través del AEE de AORTA. El AEE conoce la forma de ejecutar cualquier acción de una aplicación que está integrada con AORTA.
2. Capa de colaboración.
   - Elemento CM (*Coordination Manager*). Responsable de la coordinación. El CM gestiona políticas de coordinación relacionadas con: acceso a recursos, concurrencia, conciencia de grupo, turnos. Todas ellas orientadas a garantizar la coordinación de acciones dentro del escenario de colaboración.
   - Elemento AM (*Awareness Manager*). Responsable de la conciencia de grupo. El AM provee a los usuarios, en cada momento, información relacionada con las interacciones que suceden en el escenario de colaboración.
3. Capa de comunicación.
   - Elemento ANS (*Action Notification Service*). El ANS es el elemento responsable de notificar los eventos que acontecen en el escenario de colaboración. Las notificaciones pueden hacerse a las aplicaciones o incluso a otros elementos de AORTA.

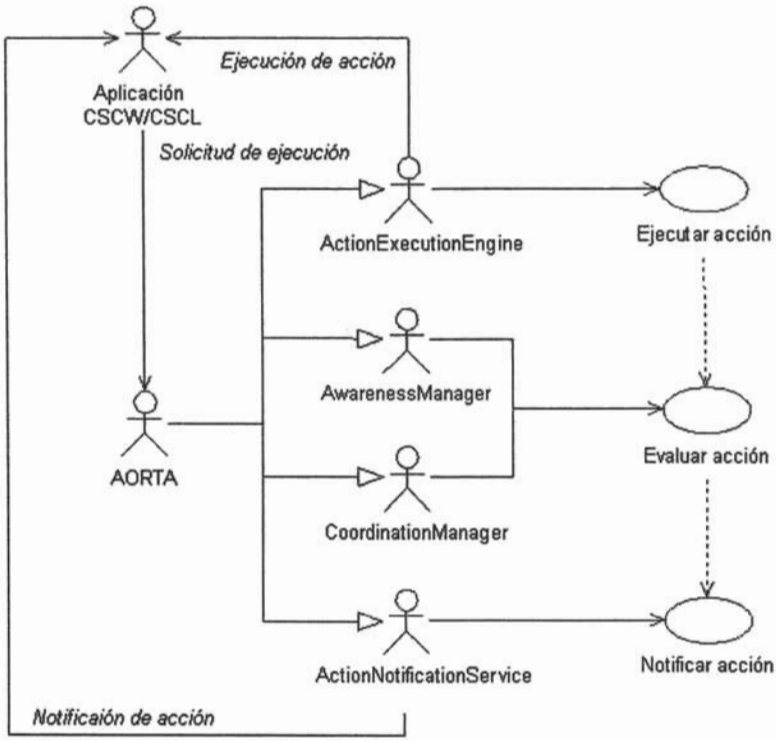


Figura 4.4: Diagrama de casos de uso de AORTA.

Esta figura muestra un diagrama con los principales casos de uso y los actores a su cargo. Como detalla este diagrama AORTA está formada por cuatro elementos. Cada elemento tiene a su cargo una tarea distinta, sin embargo, todas las tareas pertenecen al mismo objetivo: proveer servicios de coordinación a nivel de objetos y conciencia del espacio de trabajo.

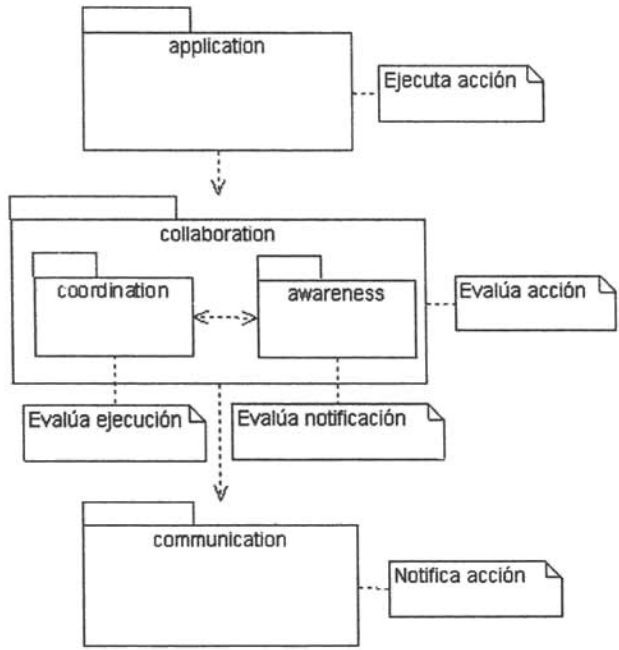


Figura 4.5: Diagrama de paquetes de las capas y subcapas de AORTA.

Esta figura describe, a través de un diagrama de paquetes, los módulos y submódulos que componen las capas de AORTA. El diagrama detalla, además, la relación que existe entre los diferentes módulos y la composición interna de cada uno de ellos.

4.2.1. Capa de aplicación

La capa de aplicación representa el punto más alto de la arquitectura y constituye, además, el puente de comunicación entre las aplicaciones y AORTA. Esta capa puede ser entendida como una capa de indirección y desacoplo debido a que provee a las aplicaciones un punto de corte en el flujo normal de su ejecución. A través de este punto AORTA incorpora, de forma desacoplada, aspectos de coordinación y conciencia de grupo en las aplicaciones.

Elementos de la capa de aplicación

Funcionalmente la capa de aplicación está conformada por un único elemento: AEE *(Action Execution Engine)*. El AEE es el elemento responsable de realizar la ejecución de las acciones de una aplicación. Las aplicaciones poseen una instancia del AEE y cada vez que la ejecución de una acción es requerida, en vez de ejecutar ellas misma la acción, solicitan al AEE la ejecución.

La relación que existe entre una aplicación y el AEE está mediada a través del uso del patrón *Command* [24] (ver figura 4.6). Este patrón, como se detalla más adelante, es una solución de diseño que permite encapsular una solicitud de ejecución dentro de un objeto desacoplándola completamente de su lógica de ejecución.

Diseño y funcionamiento de la capa de aplicación

En el caso concreto de AORTA, la idea básica en el uso del patrón *Command* es conocer, a través de la solicitud, la acción que un usuario está tratando de ejecutar antes de que ésta sea ejecutada. Desacoplar la solicitud de ejecución de la lógica de ejecución permite a AORTA parametrizar las acciones y condicionar su ejecución al cumplimiento de un conjunto de reglas (en este caso de coordinación).

Una de las ventajas más significativas en el uso de este diseño es que permite a los programadores CSCW/CSCL acceder, de forma transparente, a servicios de colaboración que son independientes a las aplicaciones.

Para interactuar con la capa de aplicación las aplicaciones CSCW/CSCL deben representar sus acciones utilizando el tipo *Action* que AORTA provee. Posteriormente, cuando la ejecución de una acción es requerida, la aplicación deberá registrarla y solicitar al AEE su ejecución (Figura 4.9/Tarea 1). El AEE evaluará la acción y responderá a la aplicación ejecutándola o emitiendo una excepción de ejecución (Figura 4.9/Tarea 5).

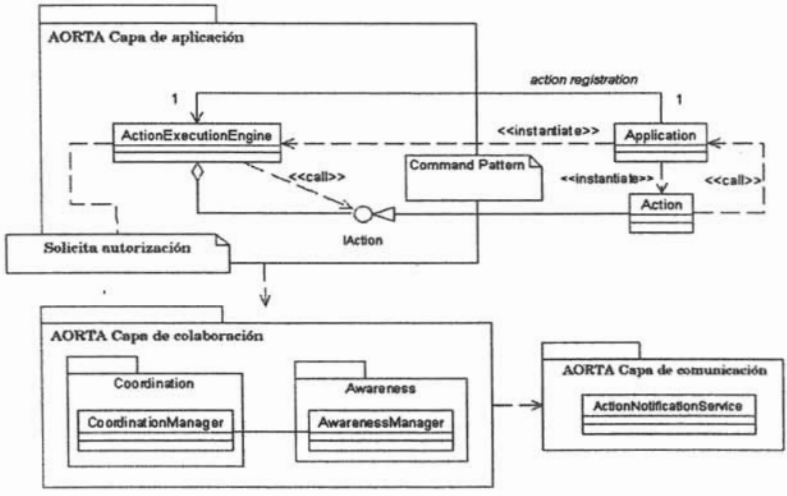


Figura 4.6: Diagrama de clases del patrón *Command* dentro de la arquitectura de AORTA.

Esta figura muestra un diagrama UML que detalla el papel del patrón *Command* y el uso de la interfaz *Action* en la relación AORTA/Aplicación. El patrón *Command* permite, por un lado, desacoplar la solicitud de ejecución de la lógica de ejecución, y por el otro, parametrizar las acciones implícitamente con servicios de colaboración. La interfaz *Action* le permite al elemento AEE conocer la forma en que deben ser ejecutadas las acciones de una aplicación particular.

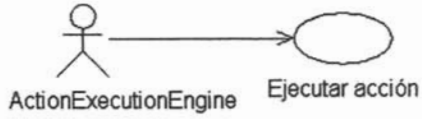


Figura 4.7: Caso de uso de la capa de aplicación.

La capa de aplicación, a través del elemento AEE, es responsable de realizar la ejecución de las acciones de una aplicación colaborativa. El elemento AEE es dependiente de la capa de colaboración. Esta última es responsable de evaluar si una acción debe o no ser ejecutada y/o notificada.

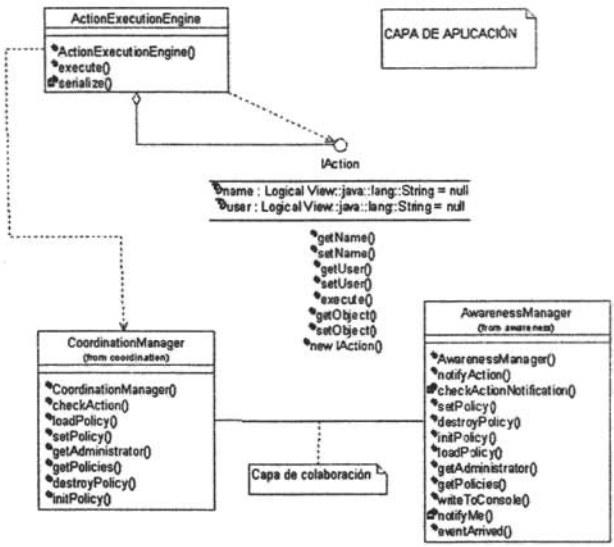


Figura 4.8: Diagrama de clases de la capa de aplicación.

Esta figura muestra un diagrama que detalla la relación entre las diferentes clases que componen la capa de aplicación y las dependencias o relaciones de éstas con clases de otras capas.

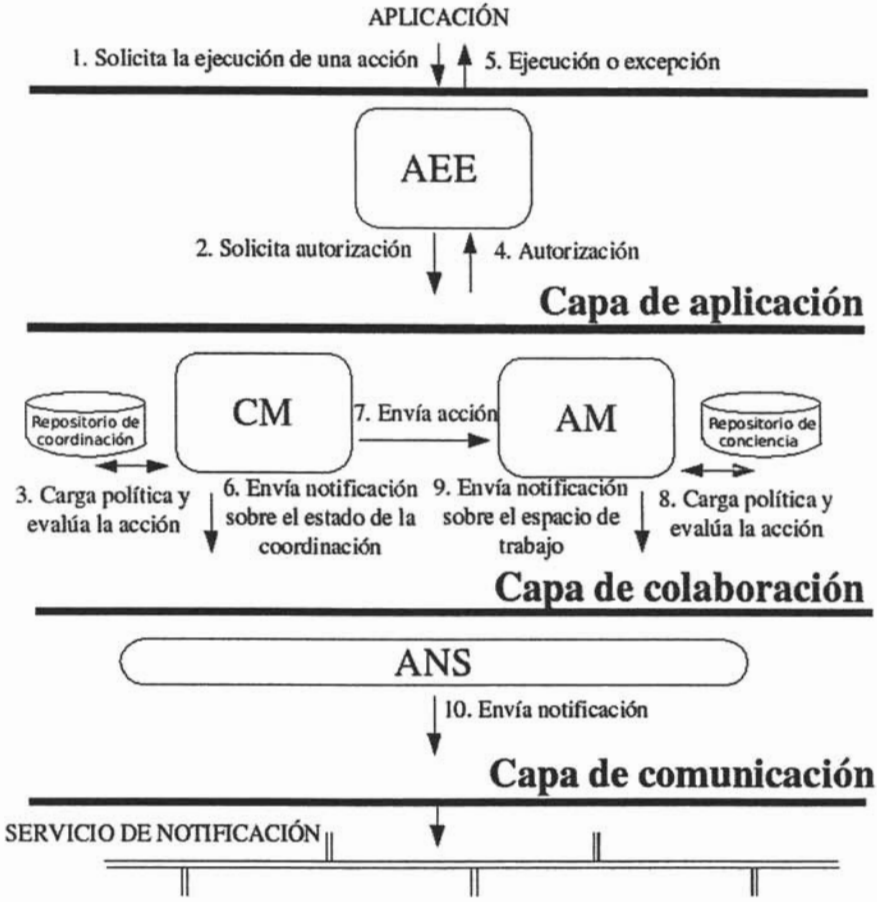


Figura 4.9: Capas y elementos de AORTA.

En esta figura se detalla la composición y el funcionamiento interno de AORTA. Se destacan la secuencia de tareas que intervienen en la ejecución de una acción, las capas donde éstas se realizan y los elementos responsables de realizar estas tareas. Esta figura es referida en repetidas ocasiones para detallar las tareas que intervienen en el proceso de ejecución de una acción.

El tipo *Action* es en realidad una interfaz que establece un contrato entre las acciones de una aplicación y el AEE de AORTA. A través de este contrato se garantiza que cualquier acción contenga la información requerida sobre: el objeto compartido que se quiere afectar, la operación se quiere realizar sobre éste y el usuario que pretende realizarla (esta información corresponde con el modelo computacional de [52][57]). Este contrato, además, le permite a AORTA conocer la forma en que debe ser ejecutada una acción.

La capa de aplicación subyace sobre la capa de servicios colaborativos. Por lo tanto, cuando el AEE recibe una solicitud para ejecutar una acción previamente registrada, éste solicita a la capa de servicios colaborativos la autorización requerida para realizar dicha ejecución. En realidad la ejecución de una acción no depende del AEE sino de la evaluación que realiza la capa de servicios colaborativos de AORTA.

4.2.2. Capa de colaboración

La capa de colaboración constituye la capa intermedia de la arquitectura de AORTA. Esta capa es responsable de ofrecer servicios de coordinación y conciencia de grupo a las aplicaciones y constituye, además, un aislamiento o desacoplo entre las capas de aplicación y comunicación.

Esta capa es proveedora de servicios de la capa de aplicación y, al mismo tiempo, consumidora de servicios de la capa de comunicación. Mientras los servicios de colaboración son demandados desde la capa de aplicación y gestionados desde la capa de colaboración los servicios de comunicación son demandados desde la capa de colaboración y gestionados desde la capa de comunicación.

Elementos de la capa de colaboración

Los servicios de colaboración, coordinación a nivel de objetos y conciencia del espacio de trabajo, se ofrecen a través de los elementos: CM *(Coordination Manager)* y AM *(Awareeness Manager)*.

El CM es el elemento responsable de realizar la toma de decisiones relacionadas con la coordinación de una aplicación. Éste evalúa si una acción puede ser ejecutada o no de acuerdo con un conjunto de reglas de coordinación que son encapsuladas dentro de un componente de software llamado política de coordinación. Si la evaluación de una acción es positiva la acción finalmente será ejecutada pero, en caso contrario, esta acción no será ejecutada y sólo será enviada una excepción de ejecución.

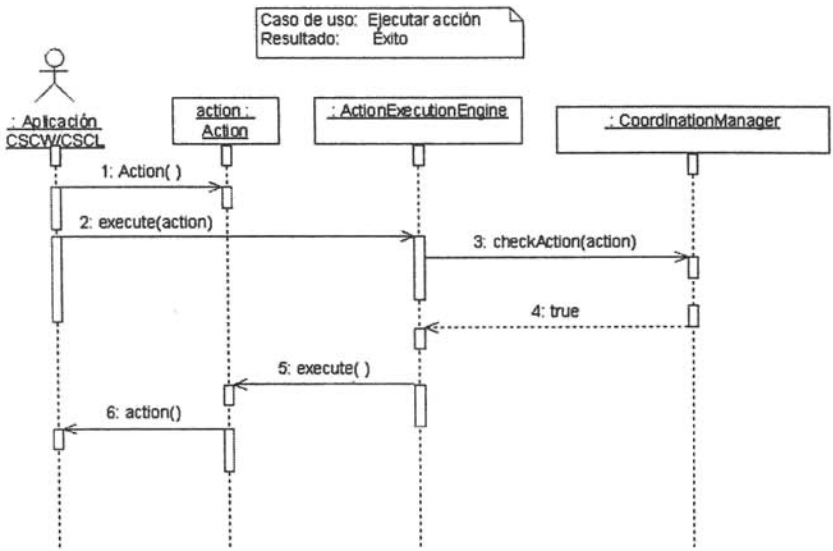


Figura 4.10: Diagrama de secuencia de la ejecución de una acción en AORTA.

Esta figura detalla la ejecución de una acción desde el momento que una aplicación hace una solicitud de ejecución al AEE. El diagrama sólo detalla la secuencia que se sigue dentro de la capa de aplicación.

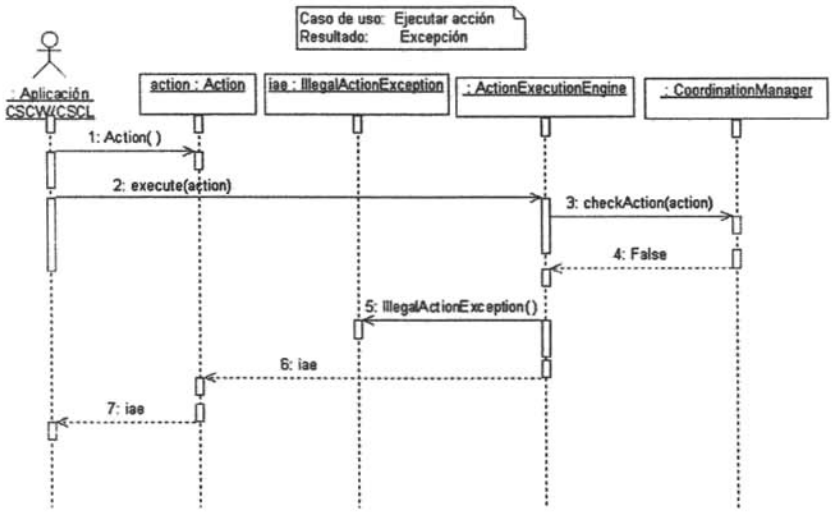


Figura 4.11: Diagrama de secuencia de la no ejecución de una acción en AORTA.

Esta figura muestra la secuencia lógica que antecede a la no ejecución de una acción. Este diagrama detalla cómo una aplicación solicita al AEE la ejecución de una acción y éste último responde la petición levantando una excepción de ejecución. Esta decisión se toma en la capa de colaboración con base en un conjunto de reglas de coordinación que han sido previamente establecidas.

Cuando la ejecución de una acción provoca un cambio en el estado de la coordinación la propagación de este cambio se da a través de una notificación de cambio de estado de coordinación. En este caso, el CM solicita a la capa de comunicación notificar a todas sus replicas sobre cambio suscitado. Estas notificaciones permiten mantener consistente el estado de la coordinación en una aplicación.

El AM es el elemento responsable de generar y administrar la información sobre el espacio de trabajo compartido. Éste recibe las decisiones de coordinación realizadas por el CM y determina si deben ser o no notificadas; en caso de serlo también decide a quienes debe notificárseles. El AM notifica, a través de la capa de comunicación, a las aplicaciones de los usuarios participantes.

El AM determina cuando y a quien debe notificar a través del uso de políticas. Las políticas de conciencia de grupo, de igual forma que las políticas de coordinación, son componentes de software que contienen instrucciones precisas sobre cuando y a quienes notificar.

El AM basa sus decisiones en las reglas que contiene la política y la información sobre la acción evaluada. Por ejemplo, el AM puede aprovechar las decisiones de coordinación para determinar cuando una acción debe ser notificada. El CM actualiza el estado de una acción con información sobre su ejecución después de evaluarla, entonces, esta información puede ser aprovechada por el AM para determinar si se notifica o no.

Diseño y funcionamiento de la capa de colaboración

La comunicación entre la capa de aplicación y la capa de colaboración se da a través de los elementos AEE (capa de aplicación) y CM (capa de colaboración) respectivamente.

El AEE solicita autorización al CM para ejecutar una acción (Figura 4.9/Tarea 2). El CM recibe la acción y evalúa su contenido a través de una política de coordinación (Figura 4.9/Tarea 3). El CM posee un repositorio de políticas y la carga de una política puede estar asociada a la ocurrencia de una acción, a un usuario determinado, a un punto en el tiempo, etc. El tipo de política puede variar entre turnos, concurrencia, conciencia de grupo, acceso a recursos, etc. Las políticas, incluso, pueden acceder a la información que se genera sobre el espacio de trabajo compartido y adaptar dinámicamente su estado en consciencia de esta información.

Una vez seleccionada y cargada la política el CM la utiliza para evaluar la acción. La política define reglas que permiten determinar si la acción es lícita o no lo es. Las reglas se basan en la información que provee una acción (acción, usuario, objeto, tiempo) y en el estado de coordinación asociado a la política.

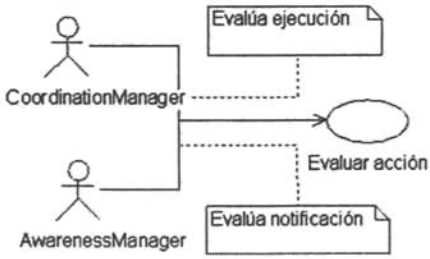


Figura 4.12: Caso de uso de la capa de colaboración.

La capa de colaboración es responsable de evaluar si una acción debe ser ejecutada y notificada. Para realizar está tarea la capa cuenta los elementos: CM (responsable de ejecución) y AM (responsable de notificación).

El estado de coordinación está dado por la secuencia de decisiones de coordinación que han sido tomadas. Este estado se almacena dentro de las políticas y se replica, si se presenta algún cambio (ver figura 4.9/Tarea 6), a las aplicaciones de cada uno de los usuarios que participan en la colaboración. Por lo tanto, en el momento de una nueva evaluación, las decisiones de coordinación se toman de forma local (con la información replicada previamente).

El CM utiliza el resultado de la evaluación para determinar si autoriza o no al AEE la ejecución de la acción y se lo notifica (Figura 4.9/Tarea 4). Si la acción ha sido autorizada el AEE la ejecuta. En caso contrario el AEE envía una excepción (Figura 4.9/Tarea 5). El CM, finalmente, actualiza la acción con el resultado de la ejecución (si se autorizó o no) y la envía al AM (Figura 4.9/Tarea 7). El AM evalúa el contenido de la acción y carga la política de conciencia de grupo correspondiente (Figura 4.9/Tarea 8). Las políticas de conciencia de grupo definen que acciones se deben notificar y a quienes. Una vez cargada la política el AM realiza las notificaciones correspondientes a través de la capa de comunicación (Figura 4.9/Tarea 8).

4.2.3. Capa de comunicación

La capa de comunicación constituye el punto más bajo de la arquitectura de AORTA y es la capa responsable de realizar todas las tareas de comunicación que se requieren. La conexión física entre una aplicación y AORTA se establece a través de está capa.

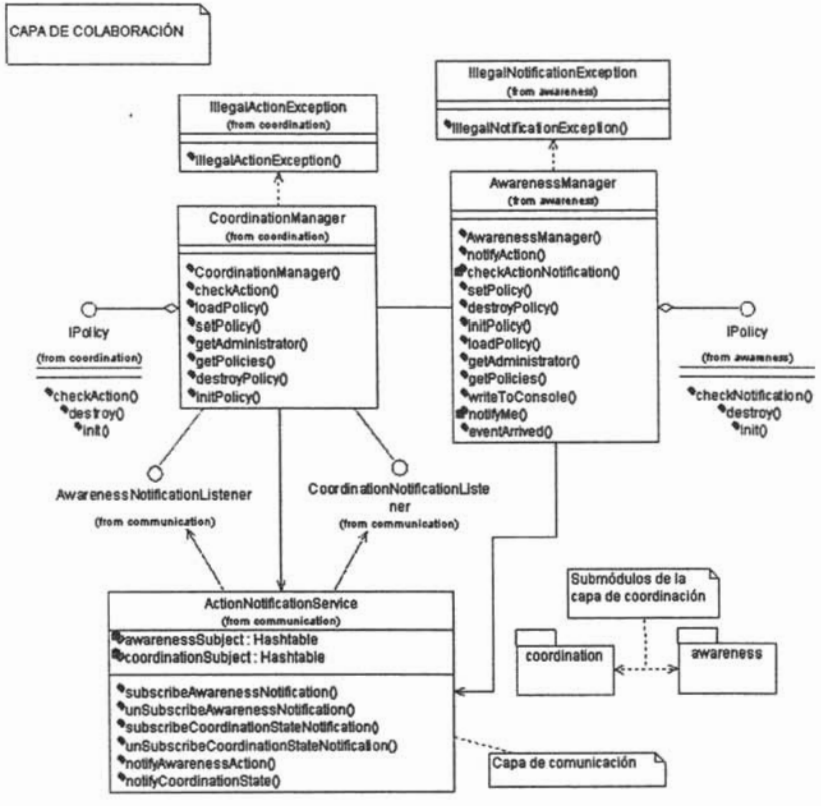


Figura 4.13: Diagrama de clases de la capa de colaboración.

Esta figura muestra el diagrama de clases de la capa de colaboración de AORTA. Este diagrama también muestra las dependencias o relaciones intercapa (capa de colaboración - capa de comunicación).

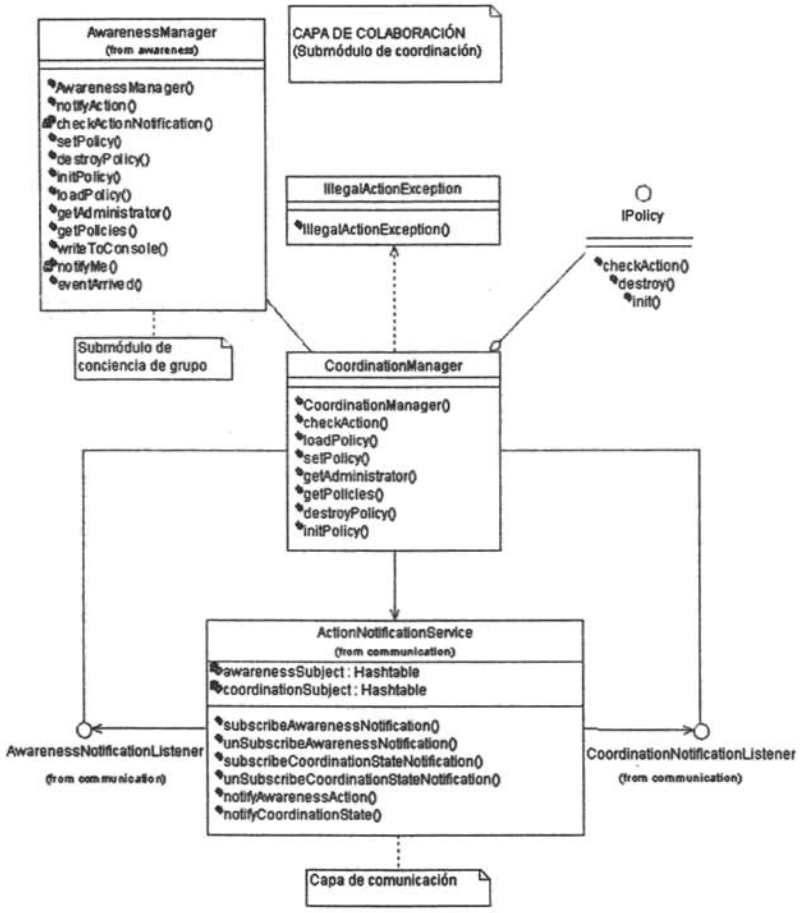


Figura 4.14: Diagrama de clases del módulo de coordinación de AORTA.

En esta figura se muestra el diseño del módulo de coordinación de AORTA mediante un diagrama de clases. El diagrama también describe la relación de las clases de este módulo con clases del módulo de conciencia de grupo o clases de la capa de comunicación. Los módulos o submódulos de coordinación y conciencia de grupo componen la capa o módulo de colaboración de AORTA.

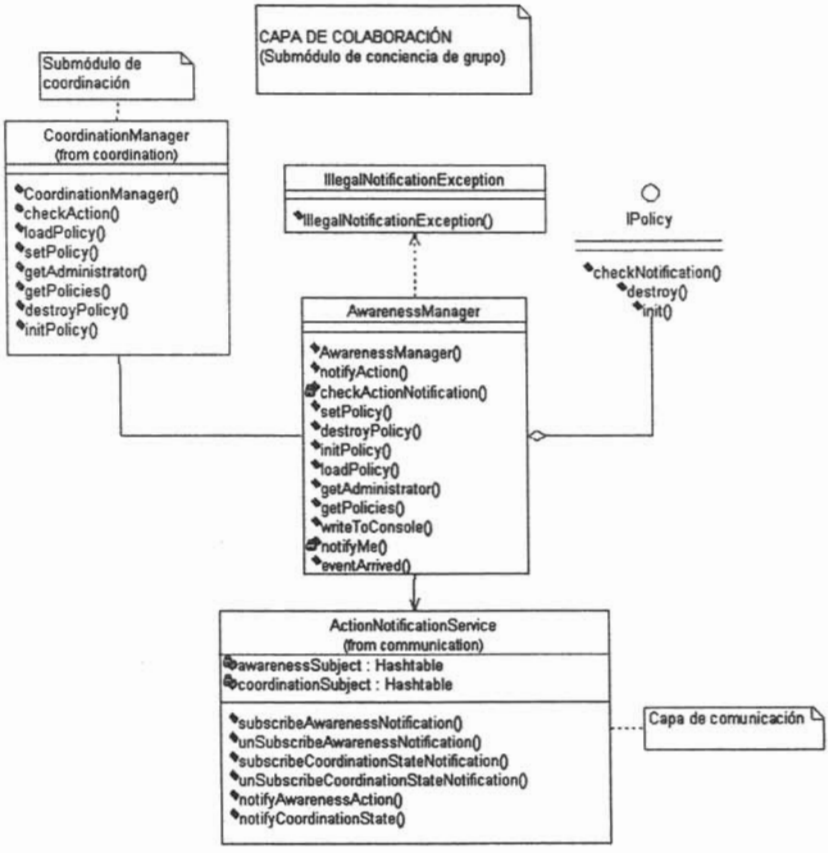


Figura 4.15: Diagrama de clases del módulo de conciencia de grupo de AORTA.

Esta figura describe el diseño de clases del módulo de conciencia de grupo de AORTA. Este diseño pone expone las relaciones y dependencias de las clases de este módulo con otros módulos (coordinación) o capas de AORTA (comunicación). Los módulos o submódulos de coordinación y conciencia de grupo componen la capa o módulo de colaboración de AORTA.

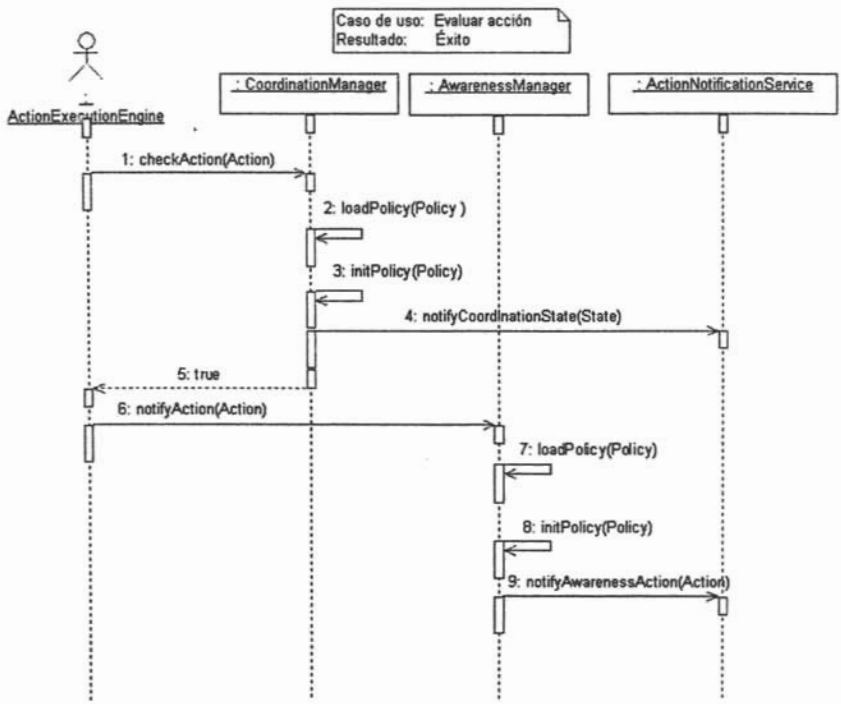


Figura 4.16: Diagrama de secuencia de la autorización de ejecución de una acción en AORTA.

Esta figura detalla la evaluación de una solicitud para la la ejecución de una acción desde el momento que el AEE solicita una autorización de ejecución al CM. El CM es el elemento responsable de evaluar si una acción debe ser ejecutada. El AEE sólo es responsable de realizar o no la ejecución.

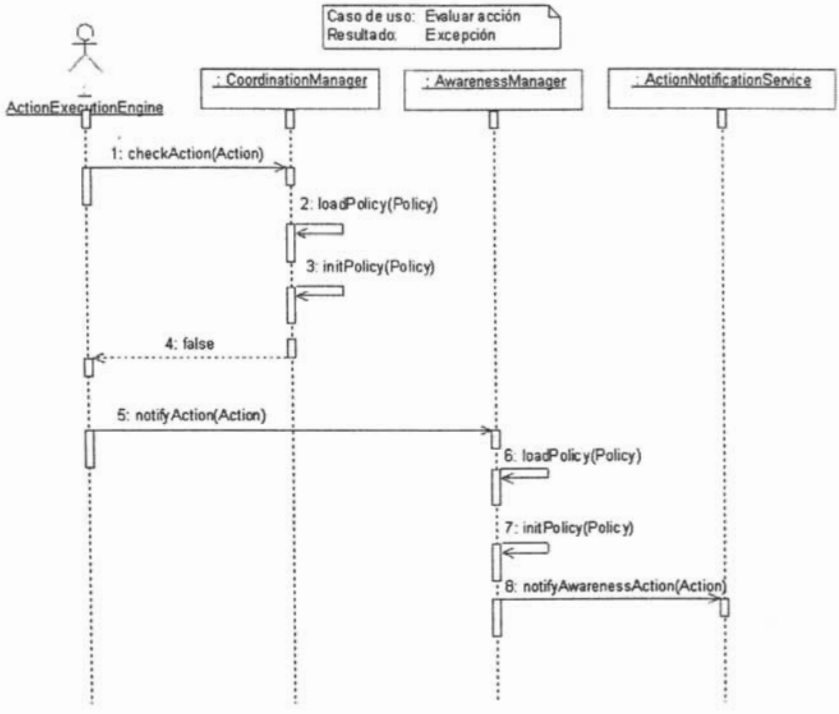


Figura 4.17: Diagrama de secuencia de la no autorización de ejecución de una acción en AORTA.

Esta figura muestra la secuencia lógica que antecede a la no autorización para la ejecución de una acción. Este diagrama detalla cómo el AEE solicita al CM autorización para la ejecución de una acción y éste último niega la autorización a partir del uso de una política de coordinación. El ejemplo presupone que la acción viola las reglas de coordinación establecidas en la política y por tal motivo la ejecución no debe realizarse.

AORTA basa fuertemente su modelo de comunicación en un servicio de distribución de eventos. Éste es un canal de comunicación, llamado bus de distribución, que transmite eventos a todas las entidades interesadas. El bus es una parte medular dentro de la arquitectura de AORTA ya que los servicios colaborativos y las comunicaciones dependen fuertemente de él.

El bus de distribución de eventos de AORTA está construido sobre un *middleware* orientado a mensajes[2] (MOM) que es realizado por terceros. Los MOM, en general, permiten modelar la interacción de componentes dentro de una arquitectura de software a través de eventos o mensajes. Los componentes son capaces de emitir eventos para advertir a otros componentes de un cambio de estado. A estos cambios los otros componentes pueden reaccionar ejecutando otras instrucciones o enviando eventos. Las ventajas de utilizar un MOM y no Llamadas a Procedimientos Remotos (RPC)[3] es que el primero permite mantener un acoplamiento menor entre los componentes, elimina las dependencias estáticas entre éstos y mejora la interoperabilidad [8][14].

Además, un MOM permite notificar eventos mediante un modelo de comunicación asíncrono a cualquier número de clientes mediante la emisión de un mismo evento. Esto agiliza notablemente el envío de datos de forma distribuida por dos razones fundamentales. La primera, al tratarse de comunicación asíncrona los procesos en ejecución tanto del lado de los clientes como del servidor no dependen del arribo de datos. Esto permite continuar la ejecución de procesos en cada uno de los actores que integran el sistema. Por ejemplo, si un cliente tarda más en recibir un evento que otro, los procesos de ejecución tanto en el servidor como en el cliente que ha recibido primero el evento, continúan su ejecución independientemente de lo que ocurra con el segundo cliente. La segunda razón es que el envío de datos a n clientes se realiza mediante la ejecución de una sola instrucción y no n como se haría con RPC.

Una de las ventajas más importantes que un modelo RPC, como RMI[4] [83], presenta sobre un MOM es que asegura que los datos enviados han llegado a su destino (algunos MOM también lo aseguran). Al ser un modelo completamente síncrono la propia ejecución del programa depende de la llegada de los datos, de otro modo e irremediablemente se tendría una excepción en tiempo de ejecución que notificaría el fallo de comunicación. Sin embargo, esto conlleva a una ejecución de procesos más lenta.

El cuadro 4.4 muestra las ventajas y desventajas entre un modelo de comunicación distribuida basado en MOM y otro basado en RPC. AORTA, particularmente, está diseñada para ofrecer servicios de coordinación y conciencia de grupo a aplicaciones colaborativas síncronas en ambientes que pueden llegar a ser altamente distribuidos. Por tal motivo su modelo de comunicación está construido sobre un MOM (Elvin [49]).

[2]Del término en inglés *Message Oriented Middleware*.
[3]Del término en inglés *Remote Procedure Call*.
[4]Del término en inglés *Remote Method Invocation*.

MOM	RPC
Los clientes son libres de realizar operaciones mientras esperan una respuesta del servidor	Provee un nivel de abstracción mayor escondiendo la lógica de distribución de los programadores
Está pensado en soportar sistemas altamente distribuidos por lo que resulta ideal para éstos	Está basado en un protocolo de comunicación de tipo solicitud/respuesta que es ideal para modelos cliente/servidor
Permite muchas respuestas para una misma solicitud o una sola respuesta para varias solicitudes	Simplifica la programación gracias a su nivel de abstracción y su modelo de comunicación solicitud/respuesta
Algunos modelos soportan prioridad, balanceo de carga, tolerancia a fallas y/o persistencia de mensajes	

Cuadro 4.4: Características distintivas de los MOM y los RPC.

Un detalle interesante de AORTA es que, gracias a la capa de comunicación, no es dependiente del MOM que utiliza (Elvin [49]). La capa de comunicación ofrece una fachada que permite acceder a diversos proveedores de MOM de forma transparente. Por lo tanto, para utilizar una comunicación segura o tolerante a fallos AORTA sólo tiene que utilizar el MOM adecuado.

Elementos de la capa de comunicación

La capa de comunicación está formada por un único elemento: ANS *Action Notification Service*. ANS es el elemento a través del cual los componentes AM y CM realizan el envío de notificaciones.

En el caso del AM las notificaciones de conciencia de grupo (interacciones que acontecen en el espacio de trabajo) se envían directamente a las aplicaciones. Mientras, en el caso del CM, las notificaciones sobre cambios de estado en el modelo de coordinación se envían a las replicas de CM de cada aplicación (los CM de cada aplicación están sincronizados a través del ANS).

Figura 4.18: Caso de uso de la capa de comunicación.

El elemento ANS es el elemento responsable de realizar todas las notificaciones (de coordinación o conciencia de grupo) que son requeridas dentro de la arquitectura distribuida de AORTA.

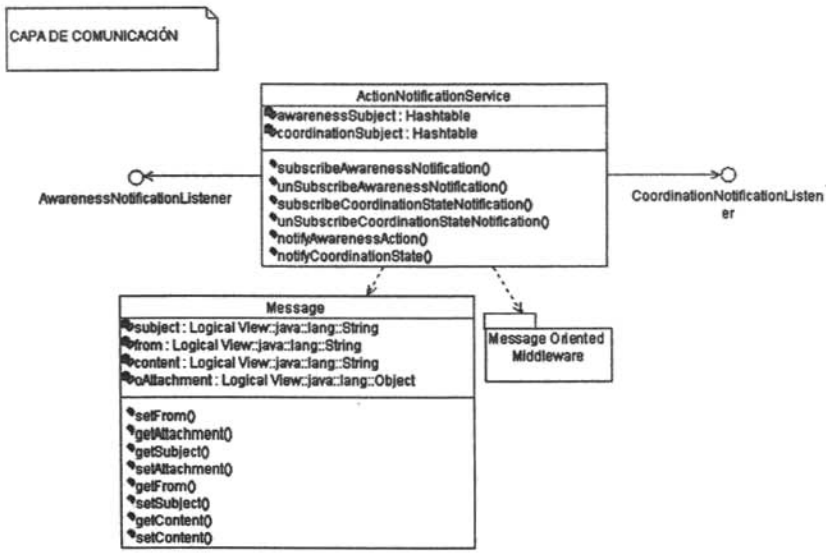


Figura 4.19: Diagrama de clases de la capa de comunicación.

Esta figura muestra un diagrama que detalla la relación entre las diferentes clases que componen la capa de comunicación y sus dependencias funcionales con el software de comunicación empleado.

Diseño y funcionamiento de la capa de comunicación

La capa de comunicación mantiene dos puntos de contacto con la capa de colaboración, determinados a través de las relaciones: CM/ANS y AM/ANS.

En el caso de la relación CM/ANS el CM accede al ANS a través de una interfaz expuesta por este último para recibir peticiones de notificación (Figura 4.9/Tarea 6). Así, cuando un CM sufre un cambio de estado solicita al ANS propagar este cambio a todas sus replicas. El ANS es realmente quien realiza la notificación de cambio de estado de coordinación a todos los CM implicados. El ANS conoce quienes son los CM involucrados porque éstos están suscritos implícitamente al servicio de notificaciones de coordinación. La suscripción a este servicio se realiza a través de la implementación de la interfaz *CoordinationNotificationListener* que AORTA implementa implícitamente en todos los CM.

El caso de la relación AM/ANS es muy similar al anterior. El AM solicita al ANS a través de una interfaz, de igual forma que el CM, la notificación de las acciones que acontecen en el espacio de trabajo compartido (Figura 4.9/Tarea 9). La diferencia más significativa con el CM es que el AM solicita que las notificaciones se realicen directamente a las aplicaciones y no a las replicas de AM de AORTA. Para recibir estas notificaciones las aplicaciones deben suscribirse explícitamente al servicio de notificación de acciones de AORTA. La suscripción a este servicio se realiza a través de la implementación de la interfaz *ActionNotificationListener* que AORTA provee a las aplicaciones.

Para realizar las notificación el ANS accede al bus de distribución de eventos. El bus de distribución de eventos es un canal de comunicación que conecta todos los puntos de la arquitectura distribuida de AORTA.

4.3. Trabajo relacionado

Existen diversos marcos para el desarrollo de aplicaciones CSCW/CSCL que ofrecen soporte de coordinación a nivel de objetos y/o conciencia del espacio de trabajo compartido. Incluso, muchos de éstos marcos ofrecen además servicios complementarios para soportar un espacio de trabajo compartido como manejo de grupos, de sesión, de objetos compartidos, etc.

Sin embargo, a diferencia de AORTA, la mayor parte de éstos marcos utilizan modelos de información basados en eventos de bajo nivel y casi ninguno de ellos ofrece un soporte integrado de coordinación a nivel de objetos y conciencia del espacio de trabajo compartido.

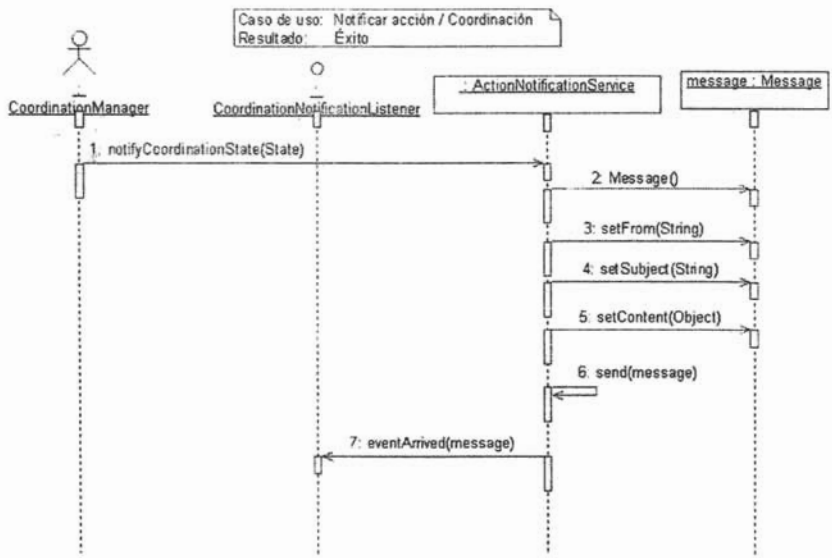


Figura 4.20: Diagrama de secuencia del envío de una notificación de coordinación.

Esta figura muestra el envío de una notificación sobre la ocurrencia de un cambio en el estado de coordinación de una aplicación. El CM es responsable de solicitar el envío de la notificación al ANS. El ANS es el elemento encargado de notificar el cambio a los diferentes puntos de la arquitectura de AORTA.

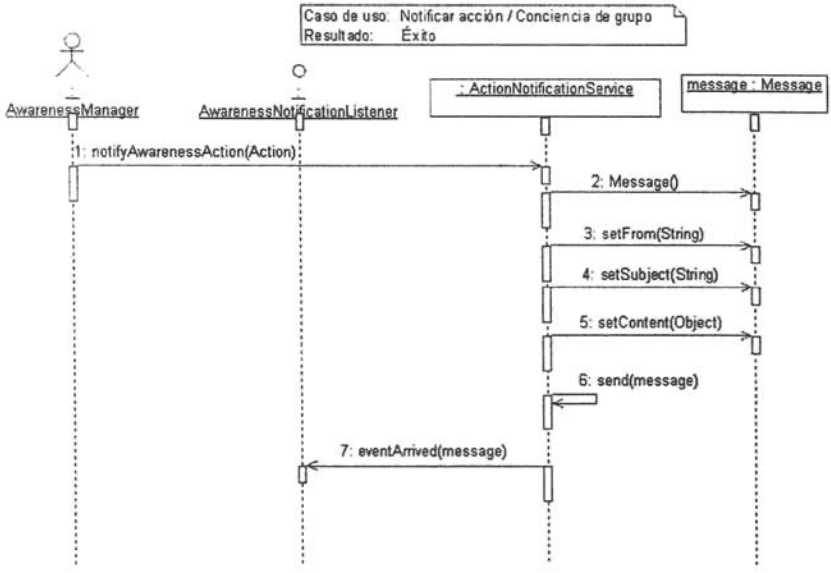


Figura 4.21: Diagrama de secuencia del envío de una notificación de conciencia de grupo.

Esta figura muestra el envío de una notificación sobre una interacción que ha ocurrido en el espacio de trabajo compartido. El AM solicitar al ANS realizar la notificación si considera que la acción realizada afecta o puede afectar el proceso de colaboración. Las reglas de evaluación para determinar si se realiza la notificación y a quien se notifica están contenidas en la política de conciencia de grupo en uso.

Dentro del ámbito del trabajo relacionado se ha realizado una selección de cinco marcos que ofrecen aproximaciones similares a la solución propuesta por AORTA. Dos de éstos, Groove y ANTS, son marcos de propósito general; uno más, GroupKit, más que un marco es un conjunto de bibliotecas también de propósito general; los dos restantes, COCA e Intermezzo, son marcos especializados en proveer soporte de coordinación a nivel de objetos.

GroupKit

GroupKit [76][77] es una herramienta clásica para el desarrollo de aplicaciones colaborativas. Sin duda, esta herramienta ha sido pionera en el desarrollo de muchas características que actualmente se mantienen vigentes en diversas arquitecturas CSCW/CSCL.

Groupkit ofrece servicios de coordinación e incluso permite la extensión de éstos servicios a través de los llamados *protocolos abiertos*. Específicamente, los protocolos abiertos se componen de tres elementos: un objeto controlado (servidor) que mantiene el estado, un objeto controlador (cliente) y un protocolo que describe cómo se comunican ambos.

Sin embargo, aunque los servicios de coordinación de Groupkit son claramente extensibles, de igual forma que en el caso de AORTA, estos servicios se centran en el uso de estructuras de datos y eventos de bajo nivel de abstracción. En el caso de la conciencia de grupo, Groupkit permite monitorizar el espacio de trabajo compartido a través de la generación de eventos. Sin embargo, el modelo de acciones de AORTA se ubica en un nivel de abstracción mayor y sus servicios de coordinación y conciencia de grupo mantienen una integración más clara de la que ambos se benefician notablemente.

ANTS

ANTS [26] basa fuertemente su arquitectura en el modelo de componentes JavaBeans y, a través de éste, ofrece una serie de servicios colaborativos que resultan muy oportunos para el desarrollo de aplicaciones CSCW/CSCL: estructuras de datos compartidas, manejo de sesiones, notificación de eventos, coordinación y conciencia de grupo. Sin embargo, en el caso específico de la coordinación y la conciencia de grupo los servicios que ANTS provee pueden resultar limitados para diversas aplicaciones colaborativas.

En el caso de la coordinación, el problema reside en que más que un servicio lo que ANTS provee son claros puntos de extensión y algunas herramientas, basadas en los *Constraint Properties* de los JavaBeans, que permiten implementar mecanismos de coordinación hechos a la medida de cada aplicación. Esto conlleva a los programadores a realizar un esfuerzo grande de desarrollo que resulta poco costeable debido a las pocas posibilidades de reutilización que tiene.

En el caso de la conciencia de grupo el modelo que ANTS provee está fuertemente basado en la ocurrencia de eventos de bajo nivel de abstracción. El problema con este servicio es que la cantidad de eventos que puede generar una aplicación es enorme y la información semántica que genera cada evento es muy pobre. Esto hace demasiado costosa la extracción de información significativa para las aplicaciones.

Groove

Groove [28] es un marco *peer to peer*[5] para el desarrollo de aplicaciones colaborativas que ofrece una extensa gama de servicios y herramientas. Diversos autores, incluso, han destacado a Groove como uno de los marcos más completos que se ha desarrollado dentro del ámbito de los sistemas colaborativos. Sin embargo, a pesar de su robusta arquitectura, Groove no provee mecanismos para el desarrollo de políticas de coordinación y su servicio de conciencia de grupo se limita a proveer información sobre la presencia y actividad de los usuarios. A diferencia de AORTA, los modelos de coordinación y conciencia de grupo de Groove no son extensibles y ambos están basados en eventos de bajo nivel de abstracción.

Intermezzo

Intermezzo [19] presenta una arquitectura cliente/servidor que ofrece servicios de colaboración a aplicaciones CSCW/CSCL. Intermezzo está relacionado con AORTA en el sentido que ambos se centran en proveer soporte de coordinación y conciencia de grupo a aplicaciones colaborativas.

A diferencia de AORTA, el soporte de coordinación que ofrece Intermezzo está basado únicamente en derechos para el control de acceso de usuarios sobre objetos compartidos. La información relacionada con las acciones que acontecen no se considera parte de la semántica que es evaluada por las reglas de coordinación. En el caso de la conciencia de grupo, aunque Intermezzo sí ofrece soporte, los servicios de coordinación jamás utilizan aprovechan está información.

COCA

COCA [44] es un marco especializado en proveer soporte de coordinación en aplicaciones colaborativas. Una de las características más interesantes de COCA es que provee un poderoso lenguaje de especificación que permite definir políticas de coordinación.

[5]El término *peer to peer*, también escrito P2P, se refiere a una red que no tiene clientes ni servidores fijos. Todos los nodos de una red P2P se comportan a la vez como clientes y como servidores de los otros nodos.

Una de las similitudes más claras entre COCA y AORTA es que ambos presentan una arquitectura distribuida que se conecta a través de un bus de colaboración. Otra similitud importante es que las políticas de coordinación en ambos casos pueden ser cargadas y cambiadas durante el tiempo de ejecución de las aplicaciones. Sin embargo, a diferencia de AORTA, COCA no ofrece ningún servicio para la conciencia de grupo. La conciencia de grupo es una parte medular dentro de la colaboración síncrona y proveer información sobre el espacio de trabajo contribuye notablemente a mejorar la coordinación [31].

4.4. Discusión

A lo largo de este capítulo se ha descrito AORTA, una arquitectura de software que ofrece soporte de coordinación a nivel de objetos y conciencia del espacio de trabajo a marcos y aplicaciones CSCW/CSCL. Dentro de las características más destacadas de AORTA se pueden mencionar las siguientes:

La integración de servicios. Los servicios de coordinación y conciencia de grupo se integran con el objeto de proveer un soporte más robusto en ambos casos (de coordinación y conciencia de grupo).

Diversos e importantes autores han destacado la fuerte dependencia que existe entre la coordinación y la conciencia de grupo. Proveer información sobre el espacio de trabajo contribuye notablemente a mejorar la coordinación [31]. La información de conciencia de grupo es siempre requerida para realizar la coordinación de actividades de grupo [16].

El uso de componentes de software. El uso de componentes de software, llamados políticas, permite extender y/o adaptar el funcionamiento de la arquitectura de acuerdo a las necesidades de cada aplicación.

Los componentes de software le permiten a AORTA ofrecer una arquitectura abierta que puede ser adaptada a diversos escenarios de colaboración o extendida por terceras partes para soportar escenarios antes no contemplados. La extensión y adaptación de AORTA no conlleva ningún cambio en las aplicaciones y se realiza de forma simple a través del cambio o inserción de componentes.

El desacoplo basado en patrones de diseño. Una de las aportaciones más importantes de este trabajo es resaltar la importancia en el uso de patrones de diseño para el desarrollo de aplicaciones CSCW/CSCL.

Aplicaciones basadas en patrones de diseño pueden ser más fácilmente integradas a marcos. Instruir a los programadores en el uso de patrones contribuye a reducir el ciclo de desarrollo de aplicaciones colaborativas.

En el caso concreto de AORTA, por ejemplo, el uso de patrones permite desacoplar la lógica que es propia de las aplicaciones de la lógica de los servicios de colaboración utilizando AOP[6] [22][63].

La orientación a acciones. La arquitectura de AORTA utiliza acciones como la unidad básica para el intercambio de información.

Proveer un modelo orientado acciones es sin duda otra de las contribuciones importantes de este trabajo ya que pone de manifiesto la necesidad de integrar modelos de información con niveles de abstracción más altos en los marcos CSCW/CSCL. Concretamente, el uso de acciones pueden aportar notables contribuciones en el desarrollo de aplicaciones CSCW/CSCL porque disminuye claramente la curva de aprendizaje de los programadores facilitando así el desarrollo o extensión de servicios de colaboración. En el caso concreto de los servicios de coordinación o conciencia de grupo, sin duda, un modelo basado en acciones es más significativo que uno basado en eventos de bajo nivel.

Durante el siguiente capítulo se describe el escenario práctico que ha sido utilizado para validar funcionalmente el primer prototipo desarrollado de AORTA. Este prototipo pone en práctica el conjunto de ideas que han sido expuestas a lo largo de este trabajo y arroja una primera estimación sobre las aportaciones e implicaciones reales que tiene el uso de AORTA para los programadores de aplicaciones CSCW/CSCL.

[6] Existen otras aproximaciones que permiten el uso de AOP utilizando software especializado como *AspectJ* [64], sin embargo, esto genera dependencias funcionales.

Capítulo 5

Prueba de concepto y validación

Con el objeto de validar el conjunto de ideas expuestas a lo largo de este trabajo se ha implementado, en el lenguaje de programación Java, un prototipo de AORTA. El prototipo ha sido integrado con un marco para el desarrollo de aplicaciones CSCW/CSCL: *ANTS framework*. Finalmente, la integración de ANTS y el prototipo de AORTA han sido validados funcionalmente a través del desarrollo de una aplicación CSCL que asiste la solución colaborativa de rompecabezas: *MagicPuzzle*.

5.1. El marco ANTS

5.1.1. Motivación

ANTS [26][25] ha sido seleccionado como parte del caso de validación de AORTA por diversos motivos. Algunos de éstos, los más importantes, se exponen a continuación:

1. ANTS es un marco para el desarrollo de aplicaciones CSCW/CSCL que provee una serie de servicios colaborativos que resultan sumamante oportunos para el desarrollo de aplicaciones CSCW/CSCL.
2. En el caso de AORTA estos servicios se vuelven aún más atractivos por tratarse de servicios distintos y complementarios a los que AORTA provee. Los servicios de ANTS y AORTA no sólo no están contrapuestos sino que funcionalmente pueden complementarse para ofrecer un soporte de colaboración más robusto.

3. El modelo de extensión que ANTS provee para terceras partes, basado en el modelo de componentes JavaBeans [87], satisface plenamente los requerimientos de AORTA.

4. ANTS ha sido desarrollado, de igual forma que AORTA, en el lenguaje de programación Java.

5. ANTS es una aplicación de código abierto[1] (su distribución incluye el código fuente) que, además, permite la modificación y libre distribución por terceras partes.

5.1.2. Diseño

ANTS es un marco que asiste a los programadores en el desarrollo de aplicaciones colaborativas. El mayor logro de diseño de ANTS es ocultar la complejidad de la lógica de sus servicios a través de un diseño basado en tres capas de software (ver figura 5.1):

- Capa de aplicaciones. Esta capa provee un contenedor para componentes JavaBeans que permite acceder de forma transparente a propiedades compartidas que son almacenadas remótamente y a un servicio de notificación de eventos.

- Capa CSCW. Esta capa ofrece los servicios básicos necesarios para soportar un espacio de trabajo compartido: soporte de sesiones, soporte de objetos compartidos, soporte básico de coordinación y soporte básico conciencia grupo.

- Capa de tecnología. Esta capa está constituida fundamentalmente por un bus de notificación de eventos que ha sido construido sobre un servicio de mensajería (realizado por terceras partes).

5.1.3. Servicios

Sesiones compartidas. El sistema de gestión de sesiones de ANTS es un modelo basado en tres jerarquías: *World, Place y Thing.*

Un componente *World* representa la jerarquía más alta. Éste constituye un contenedor o envolvente para un grupo de sesiones. Las sesiones colaborativas dentro del modelo de ANTS son llamadas *Places*, mientras que las aplicaciones con las que se podrá colaborar en cada sesión reciben el nombre de *Things.*

[1] Del término en inglés *Open Source*. Para más información: http://www.opensource.org.

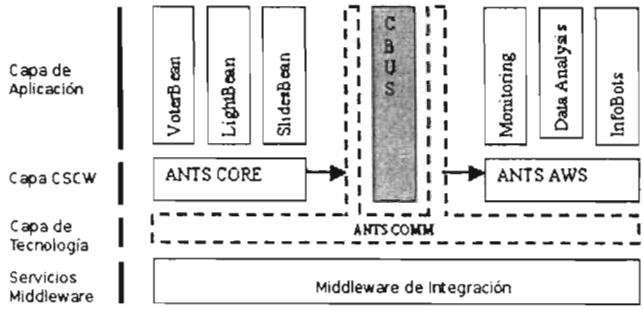


Figura 5.1: Arquitectura de ANTS *framework*.

Esta figura muestra la arquitectura en capas de ANTS. Las aplicaciones que residen en ANTS subyacen sobre la capa de aplicaciones que es la capa más alta. Está capa subyace sobre la capa CSCW que es donde está contenida toda la lógica de los servicios de colaboración que ANTS ofrece. La capa más baja de la arquitectura es la capa de tecnología. Esta capa está constituida fundamentalmente por un bus de notificación de eventos que ha sido construido sobre un MOM (realizado por terceras partes).

El modelo de sesiones de ANTS es rígido: si una jerarquía de tres niveles no resulta suficiente para soportar un escenario de colaboración no es posible extender el modelo. Otra característica importante del modelo de sesiones de ANTS es que éste se maneja de forma centralizada, esto quiere decir que sus propiedades se almacenan en un servidor central dedicado a la persistencia de datos.

Soporte de coordinación. La coordinación en ANTS se gestiona de forma centralizada haciendo uso del concepto de *Token*: solamente quien tenga el *Token* está autorizado para hacer cambios en las propiedades.

Para implementar cualquier otro tipo de política ANTS provee a los programadores un modelo para el desarrollo de propiedades compartidas a través del cual permite la creación de mecanismos de coordinación hechos a la medida de cada aplicación. El modelo de propiedades compartidas de ANTS es una extensión del modelo *Constraint Properties* de los JavaBeans.

Lo que hacen los *Constraint Properties* es permitir la definición de intermediarios capaces de interceptar cualquier petición de cambio sobre una propiedad compartida. Los intermediarios o Coordinadores, como ANTS los llama, pueden responder al intento de cambio de una propiedad compartida emitiendo una excepción que finalmente evita que el cambio se realice.

La lógica que contiene cada Coordinador para determinar cuando y de qué forma debe aceptarse el cambio a una propiedad compartida es lo que ANTS llama política de coordinación. El desarrollo de políticas de coordinación en ANTS se hace a la medida de cada aplicación y es, además, responsabilidad de cada programador desarrollarla.

Soporte de conciencia de grupo. La conciencia de grupo en ANTS se basa en un servicio de monitorización de eventos de bajo nivel de abstracción. Este servicio permite asociar la ejecución de procesos a la aparición de algún evento específico o, incluso, a la llegada de algún punto cualquira en el tiempo. El bus de colaboración de ANTS es parte fundamental en este proceso. Al ser el bus el elemento encargado de propagar cualquier evento que pueda originar un cambio en el estado de la colaboración éste es también responsable de propagar esos mismos eventos hasta el servicio de monitorización de ANTS.

De igual forma que en el caso de la coordinación, ANTS permite que sean los programadores quienes desarrollen, a la medida de sus aplicaciones, los procesos que serán asociados a la ocurrencia de eventos.

5.1.4. Evaluación

En términos generales se puede decir que la utilización de ANTS como una herramienta de desarrollo, como la de cualquier otro marco, conlleva un costo de aprendizaje alto. Utilizar una herramienta de estas características requiere un conocimiento basto que necesitará invertir varias horas en su aprendizaje.

Sin embargo, en el caso de ANTS, la inversión de tiempo es redituable porque, una vez superada la curva de aprendizaje, permitirá ahorrar mucho trabajo de desarrollo a los programadores. Sin duda, el mayor logro de ANTS es ocultar a los programadores toda la complejidad de sus servicios exponeniéndolos de una forma simple y abstracta. Un claro ejemplo de esto es el servicio de de sesiones compartidas, posiblemente la parte más robusta de ANTS. El manejo de sesiones compartidas provee a los programadores en una forma abstracta todo el soporte que requieren para soportar un espacio de trabajo compartido.

Otro de los aspectos más destacados de ANTS es la portabilidad de su modelo de comunicación. El módelo de comunicación de ANTS está construido sobre un MOM. Sin embargo, ANTS utiliza una capa de software intermedia que lo aisla del MOM utilizado. ANTS puede acceder, de forma transparente, a diferentes proveedores de MOM.

Sin duda, el modelo de coordinación de ANTS es uno de sus aspectos más débiles. La política de coordinación de todo o nada, como se puede llamar al *Token*, puede resultar efectiva en muchos casos, sin embargo, para la mayor parte de las aplicaciones colaborativas resulta sumamente limitado restringirse a este modelo de coordinación.

Proporcionar una sola política de coordinación no parece suficiente. Como solución a este problema, ANTS permite que su modelo de coordinación sea extendido por terceras partes. Sin embargo, desarrollar políticas de coordinación hechas a la medida de cada aplicación es una tarea difícil que requiere un profundo conocimiento del marco y que tiene pocas posibilidades de ser reutilizado en otras aplicaciones. Por lo que tampoco parece una solución ideal.

En términos generales, basar el modelo de coordinación de cualquier aplicación colaborativa en las propiedades *Constraint Properties* de los JavaBeans no parece oportuno. Aplicaciones colaborativas síncronas, donde el tiempo de respuesta es vital y el número de interacciones que se generan es considerablemente grande, no parecen ajustarse a un modelo de coordinación de estas características.

Otro problema evidente en ANTS es su servicio de conciencia de grupo. Este servicio se basa en la monitorización de los eventos que se generan en el espacio de trabajo compartido. El problema reside en que, al tratrse de eventos de tan bajo nivel de abstracción, la cantidad de eventos que puede generar una aplicación es enorme, la información semántica que aporta cada evento es muy poca y, finalmente, procesar esa cantidad de información de forma útil es una tarea muy complicada.

5.2. MagicPuzzle

MagicPuzzle (MP) es una aplicación síncrona que asiste en la solución colaborativa de rompecabezas a grupos de estudiantes de educación primaria. En general, el desarrollo de aplicaciones CSCL síncronas de juegos de piezas está fuertemente respaldado por los beneficios que éstos ofrecen en materia de educación, de socialización y, además, por su capacidad para reflejar el proceso de construcción de conocimiento.

En el caso específico de este trabajo se ha desarrollado un prototipo de MP con el propósito de validar funcionalmente la integración AORTA/ANTS. Por lo tanto, este prototipo cuenta simultaneamente con dos proveedores de servicios a los que accede de forma transparente: mientras ANTS provee servicios de sesión, objetos compartidos y grupos, AORTA provee los servicios de coordinación y conciencia de grupo (ver figuras 5.2 y 5.3).

Como se detalla a continuación, el uso de este marco híbrido en el desarrollo de una aplicación CSCL ha permitido, por un lado, llevar a la práctica el conjunto de ideas expuestas a lo largo de este trabajo y, por el otro, evidenciar las implicaciones que estas mismas ideas han tenido en el ciclo de desarrollo de dicha aplicación.

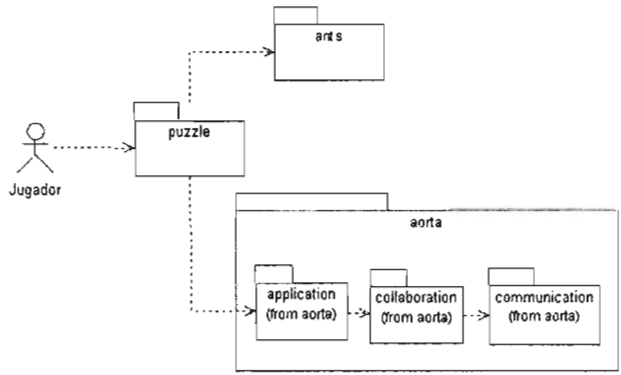


Figura 5.2: Diagrama de paquetes: MagicPuzzle/AORTA/ANTS.

Esta figura detalla, mediante un diagrama de paquetes, la relación MagicPuzzle/AORTA/ANTS. MagicPuzzle depende funcionalmente de ANTS, porque le provee servicios de sesión, objetos compartidos y grupos; y de AORTA porque le provee servicios de coordinación y conciencia del espacio de trabajo.

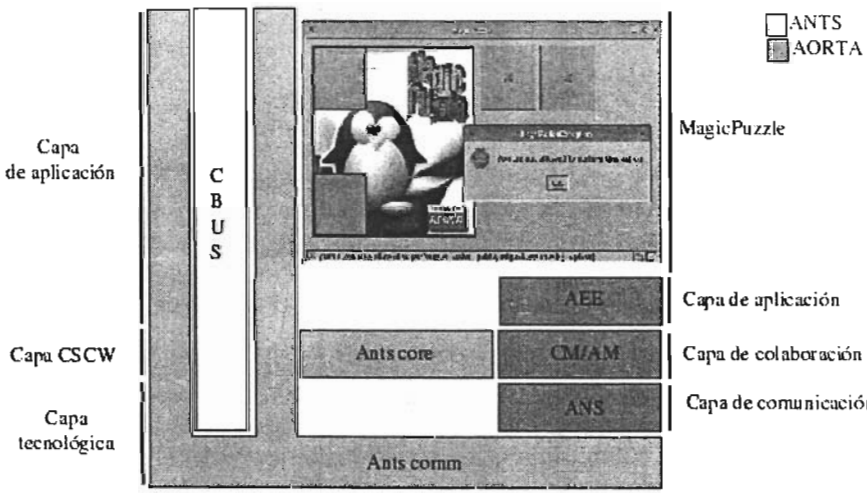


Figura 5.3: Diagrama de bloques: MagicPuzzle/AORTA/ANTS.

Esta figura muestra a través de un diagrama de bloques la relación MagicPuzzle/AORTA/ANTS. Como se muestra en el diagrama MagicPuzzle ha sido construido sobre el marco híbrido resultante de la integración AORTA/ANTS.

5.2.1. Adecuación

AORTA es una arquitectura orientada acciones, por lo tanto, para su utilización, MP ha tenido en cuenta las siguientes consideraciones de diseño:

- Identificar las acciones que forman parte de la aplicación. En este caso, por tratarse de un primer prototipo, se ha decidido utilizar sólo dos acciones: seleccionar pieza y soltar pieza.
- Encapsular la lógica que interviene en cada una de las acciones. En MP la lógica referente a las acciones seleccionar pieza y soltar pieza ha sido encapsulada a través de las clases *SelectPiece* y *DropPiece* respectivamente. *SelectPiece* contiene la lógica que permite a los participantes seleccionar y modificar la posición de una pieza mientras que *DropPiece* contiene la lógica que permite liberar la pieza previamente seleccionada (ver figura 5.4, clases *SelectPiece* y *DropPiece*).
- Ejecutar la lógica de las acciones a través del elemento AEE de AORTA (ver figuras 5.5, 5.6).

A través de estás modificaciones ha sido posible para MP acceder a los servicios de coordinación de acciones y conciencia del espacio de trabajo que ofrece AORTA. A continuación se describe la funcionalidad específica de cada uno de los servicios incorporados a MP.

5.2.2. Servicios de colaboración que AORTA provee

Política de coordinación *Token*. Esta política restringe accesos concurrentes a objetos compartidos. La política se encarga de registrar el usuario que inicia una acción y el objeto sobre el que dicha acción se realiza.

En el caso de MP las piezas representan los objetos compartidos. Cuando una pieza se selecciona, a través de la acción *SelectPiece*, la política registra la pieza y bloquea accesos posteriores a ésta. La pieza se libera una vez que la acción *DropPiece* es ejecutada.

La política *Token*, además, utiliza la información que se genera sobre el espacio de trabajo para actualizar su estado dinámicamente: si un usuario intenta realizar una acción ilegal cinco veces o más la política lo bloquea permanentemente. Los usuarios conocen que piezas están deshabilitadas porque MP utiliza la información de conciencia de grupo que AORTA provee para actualizar su interfaz. Por ejemplo, una pieza seleccionada por otro usuario es marcada con una cruz y los usuarios que están realizando acciones son mostrados junto con estás en la barra de estado de MP (ver figura 5.7).

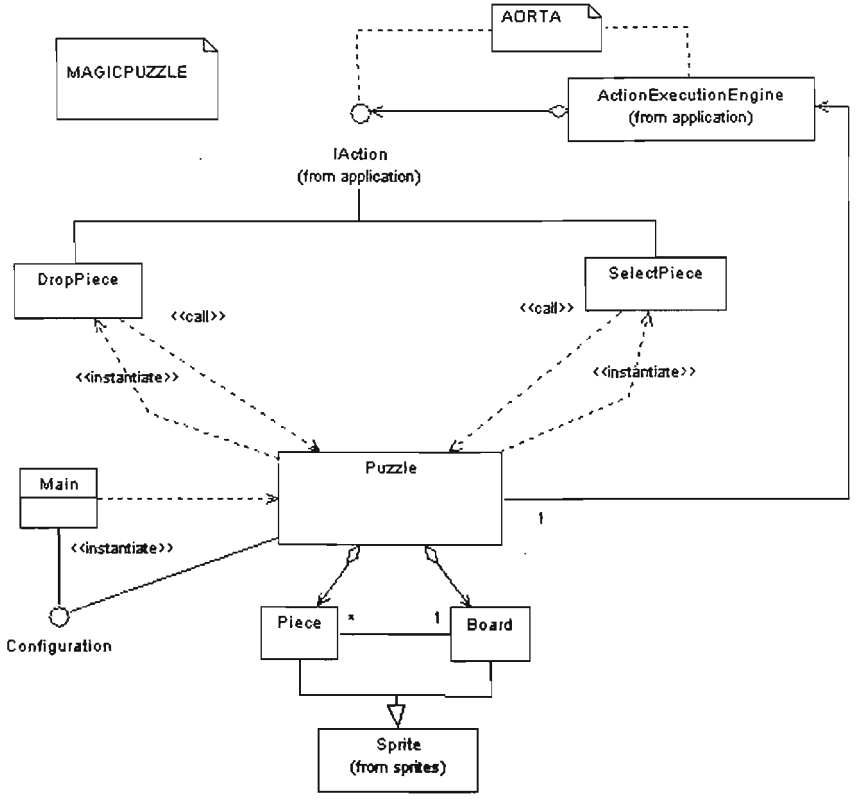


Figura 5.4: Diagrama de clases de MagicPuzzle.

El diagrama de clases que se muestra en esta figura detalla parte del diseño de la aplicación MagicPuzzle. Como se puede apreciar en la parte más alta del diagrama MagicPuzzle mantiene una asociación directa con el elemento *ActionExecutionEngine* (de AORTA) y otra más con la interfaz *IAction* (de AORTA) que se produce a través de sus clases *SelecPiece* y *DropPiece*.

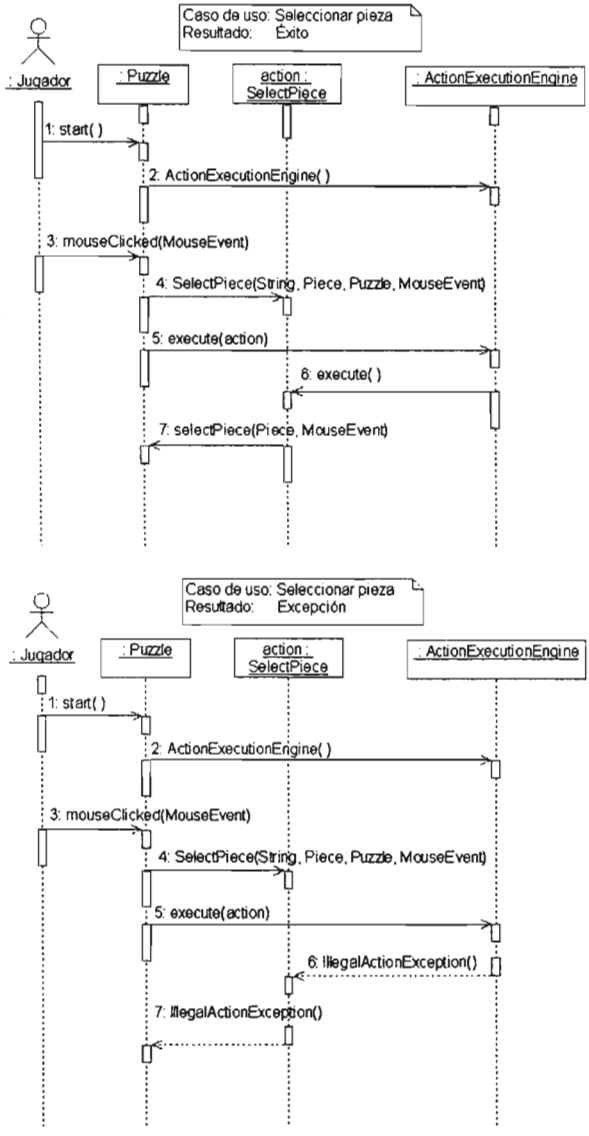


Figura 5.5: Diagramas de secuencia del caso de uso *Seleccionar Pieza* en MagicPuzzle.

Esta figura muestra dos diagramas de secuencia del caso de uso *Seleccionar Pieza* de MagicPuzzle. En el primer diagrama la secuencia finaliza con la ejecución de la acción *SelectPiece* mientras en el segundo diagrama la secuencia finaliza emitiendo la excepción *IllegalActionException*.

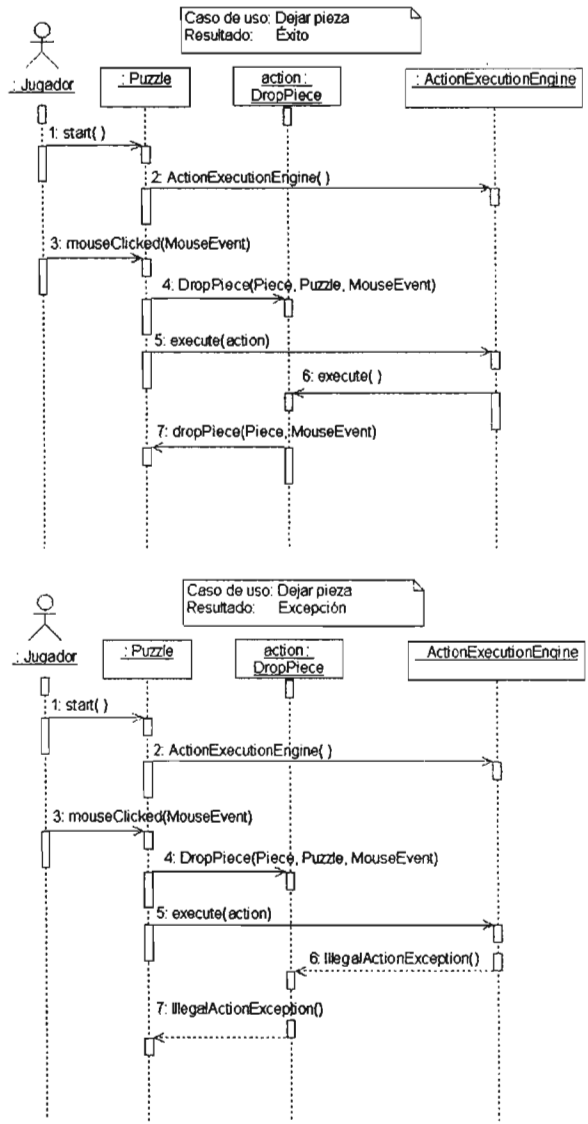


Figura 5.6: Diagramas de secuencia del caso de uso *Dejar Pieza* en MagicPuzzle.

Esta figura muestra dos diagramas de secuencia del caso de uso *Dejar Pieza* de MagicPuzzle. En el primer diagrama la secuencia finaliza con la ejecución de la acción *DropPiece* mientras en el segundo diagrama la secuencia finaliza emitiendo la excepción *IllegalActionException*.

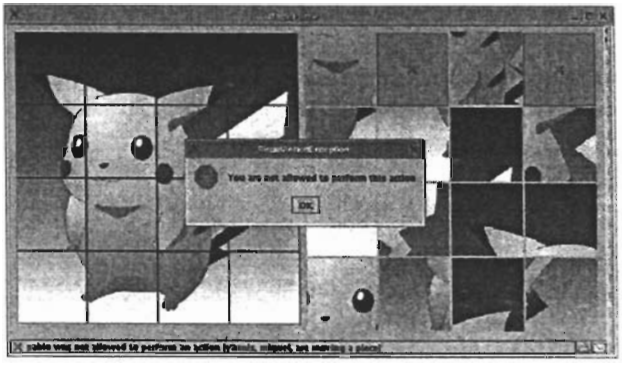


Figura 5.7: Política *Token* en MagicPuzzle (escena 1).

Esta figura muestra una escena del juego MagicPuzzle donde se exhibe el intento de un usuario por realizar una acción ilegal: seleccionar una pieza previamente seleccionada por otro usuario.

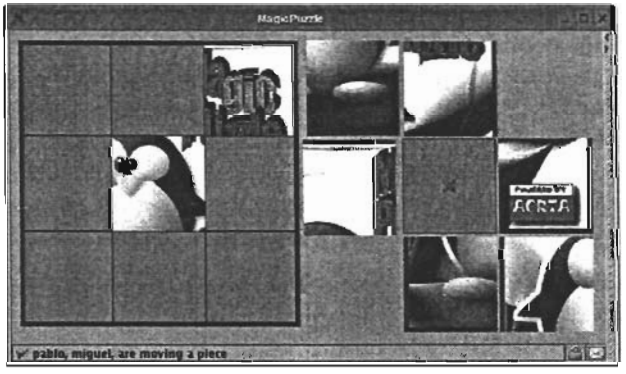


Figura 5.8: Política *Token* en MagicPuzzle (escena 2).

Esta figura muestra una escena del juego MagicPuzzle donde diversos usuarios realizan movimientos de pieza de forma simultánea. La información sobre los movimientos se muestra en la barra de estado de MagicPuzzle

Política de coordinación *Leader*. Esta política recibe como datos de entrada una lista de participantes y un líder. La política muestra en cada turno la lista de participantes al líder y éste selecciona al que corresponde el siguiente turno. En el caso de MP, sólo el participante seleccionado puede realizar la acción *SelectPiece* y la política pregunta al líder por un nuevo participante cada vez que la acción *DropPiece* se realiza (ver figura 5.9).

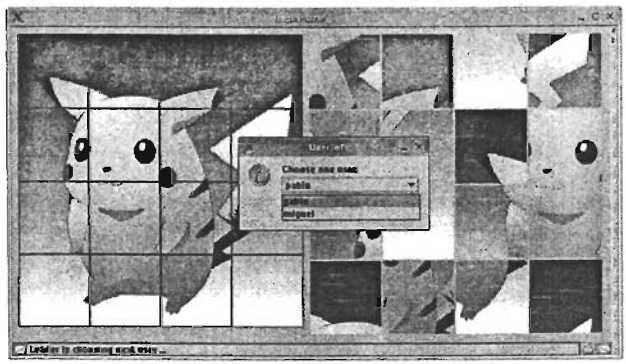


Figura 5.9: Política *Leader* en MagicPuzzle.

Esta figura muestra una escena del juego MagicPuzzle realizado con la política *Leader*. El participante identificado como líder está seleccionando a que participante corresponde el siguiente turno.

Política de coordinación *Turn*. Esta política recibe como datos de entrada la lista de participantes. La política no permite a los participante realizar la acción *SelectPiece* de forma consecutiva. Por ejemplo, para realizar dos veces o más la acción *SelectPiece* un participante deberá esperar que el resto de participantes realicen la misma acción igual número de veces. MP indica a que usuario corresponde el turno en cada momento a través de su barra de estado (ver figura 5.10).

Política de conciencia de grupo *Everybody*. La política *EveryBody* notifica, al momento, el acontecimiento de todas las acciones a todos los participantes. En el caso de MP, cada vez que los participantes tratan de realizar la acción *SelectPiece* (aunque la acción no se realice por restricciones del modelo de coordinación) el intento y su resultado es notificado al resto de los participantes. Variantes de esta misma política podrían notificar sólo acciones que sí se han realizado o que han sido realizado por un determinado usuario.

Política de conciencia de grupo *Leader*. La política recibe como datos de entrada una lista de participantes y un líder. Esta política es responsable de notificar el acontecimiento de todas las acciones que se pretenden realizar junto con su resultado al líder (ver figura 5.11). A diferencia de la política *Everybody* participantes sin el rol de líder no son notificados.

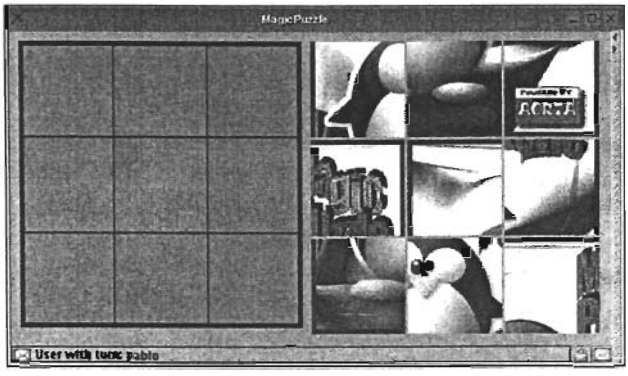


Figura 5.10: Política *Turn* en MagicPuzzle.

Esta figura muestra una escena del juego MagicPuzzle realizado con la política *Turn*. El participante al que corresponde el turno está siendo mostrado en la barra de estado de MagicPuzzle.

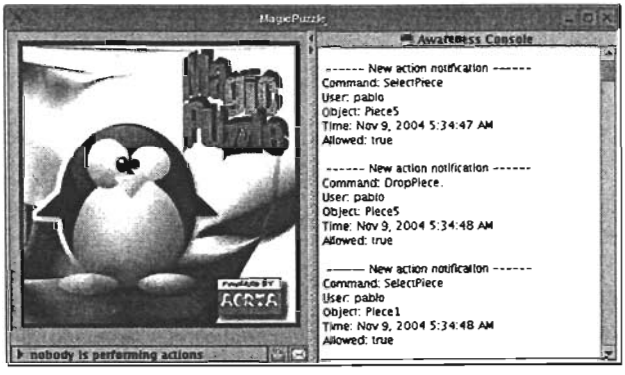


Figura 5.11: Consola de información de MagicPuzzle.

Esta figura muestra una escena del juego MagicPuzzle realizado con la política de conciencia de grupo *Leader*. El participante identificado como líder ha recibido notificaciones sobre todas las acciones realizadas por los participantes. Las notificaciones se muestran a través de la consola de información del espacio de trabajo de MagicPuzzle.

Cambios de política en tiempo de ejecución. Las políticas de coordinación y conciencia de grupo de MagicPuzzle pueden ser cambiadas durante el tiempo de ejecución (ver figura 5.12). Gracias a este hecho para MagicPuzzle es posible manejar escenarios complejos a través del uso de dos o más políticas distintas durante el mismo juego.

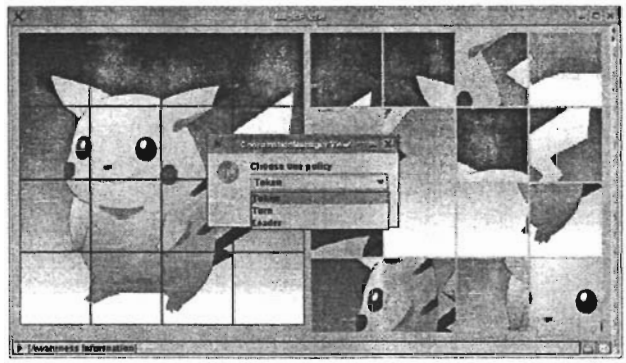


Figura 5.12: Cambio de política en tiempo de ejecución en MagicPuzzle.

Esta figura muestra una escena del juego MagicPuzzle en la que se está realizando un cambio de política de coordinación en el tiempo de ejecución de la aplicación.

5.3. Discusión

AORTA ha permitido a MagicPuzzle beneficiarse con servicios robustos de coordinación y conciencia de grupo. Estos servicios están basados en un modelo de componentes de software, llamados políticas, que ha permitido que MagicPuzzle pueda adecuar su soporte de coordinación y conciencia de grupo a diversos escenarios de colaboración. La adecuación de MagicPuzzle no conlleva ningún cambio (se realiza de forma transparente a los programadores) y puede sucitarse incluso durante el tiempo de ejecución de la aplicación.

AORTA, además, le da la posibilidad a MagicPuzzle de extender su soporte de coordinación y conciencia de grupo a escenarios de colaboración no contemplados anteriormente. El soporte que AORTA ofrece puede ser extendido por terceras partes a través del desarrollo e inclusión de nuevas políticas. El uso de nuevas políticas es horizontal aun para aplicaciones existentes como MagicPuzzle. Sin embargo, debido a que AORTA no cuenta con un lenguaje de especificación de políticas el desarrollo de éstas debe, necesariamente, realizarse por programadores.

Otro de los beneficios palpables para MagicPuzzle ha sido la integración de servicios de colaboración de forma desacoplada. AORTA permite tener una clara separación entre la lógica de aplicación y la lógica de colaboración que se ha traducido en obvios beneficios de desarrollo y mantenimiento para los programadores de MagicPuzzle.

Sin duda, uno de los beneficios más notables que ha traído consigo AORTA es permitir a los programadores trabajar a nivel de acciones. El servicio de notificación de AORTA está basado en acciones y no en eventos. La extensión de los modelos de coordinación y conciencia de grupo de AORTA se hace a través del desarrollo de nuevas políticas, orientadas al manejo de acciones y no de eventos.

Las acciones son un concepto más cercano a los programadores de aplicaciones CSCW/CSCL que los eventos y esta aseveración ha quedado de manifiesto en el desarrollo de MagicPuzzle. Por ejemplo, para los programadores de MagicPuzzle recibir acciones y generar la lógica necesaria para traducirlas en actualizaciones de la vista o el modelo ha sido una tarea más simple que generar esos mismos cambios a partir de eventos. De igual forma, generar nuevas políticas a partir de acciones ha resultado una tarea más simple para los programadores que generar esas mismas políticas a partir de eventos.

Finalmente es importante mencionar que el uso de acciones en MagicPuzzle también ha tenido implicaciones de diseño. En consecuencia, aunque el proceso de adecuación ha sido simple, trabajar a nivel de acciones sí requirió un esfuerzo adicional por parte de los programadores.

Capítulo 6

Conclusiones y trabajo futuro

6.1. Conclusiones

A lo largo de este trabajo de tesis se ha presentado AORTA [61], una arquitectura de software basada en componentes que ofrece soporte de coordinación a nivel de objetos y soporte de conciencia del espacio de trabajo a marcos y aplicaciones CSCW/CSCL síncronas.

Sobre las aportaciones más destacadas que ha hecho este trabajo a través del diseño, implementación y validación de AORTA es posible concluir lo siguiente:

Arquitectura abierta

Los escenarios colaborativos se caracterizan por ser entornos altamente dinámicos e impredecibles. En consecuencia, casi cualquier aplicación que pretenda asistir la colaboración deberá estar preparada para enfrentar un ciclo evolutivo intenso. Preparar una aplicación significa proveerle mecanismos que le permitan adaptarse y extenderse de forma sistemática.

En el caso de AORTA diseñar una arquitectura de fácil extensión y adaptación ha sido una prioridad. Apoyándose en el uso de componentes de software AORTA ofrece una arquitectura abierta que puede ser adaptada o extendida de forma simple a diversos escenarios de colaboración.

La adaptación de AORTA de un escenario a otro se da a través de cambios de políticas. Las políticas son componentes de software que determinan el soporte específico de coordinación y conciencia de grupo con que cuenta una aplicación. A través de cambios en sus políticas, que pueden suscitarse incluso en tiempo de ejecución, las aplicaciones que utilizan AORTA pueden adaptar o modificar su modelo de coordinación y conciencia de grupo sin necesidad de efectuar ninguna modificación. Incluso, en algunos casos, el uso de estas políticas es discrecional para las aplicaciones.

AORTA es una arquitectura de software extensible. La extensión de AORTA se produce a través del desarrollo e inclusión de nuevas políticas. AORTA presenta claros puntos de extensión y nuevas políticas pueden ser desarrolladas y adicionadas por terceras partes. Debido a que AORTA ofrece un modelo de componentes para el desarrollo de nuevas políticas el uso de éstas es transparente para cualquier aplicación, incluso para aquellas que han sido desarrolladas previamente.

Entre los faltantes más destacados en el modelo de políticas que ofrece AORTA se tiene la falta de un lenguaje de especificación de políticas. Debido a esto el desarrollo de nuevas políticas es una tarea que debe ser realizada necesariamente por programadores. Otro problema notorio con el desarrollo de políticas es su reutilización. Problema largamente citado dentro del ámbito de CBSE y que ha quedado de manifiesto en el uso de AORTA: cuanto más especializadas son las políticas sus posibilidades de ser reutilizadas son menores.

Sin embargo, a pesar de estos problemas, el modelo de políticas de AORTA ha mostrado ser una herramienta útil que permite a las aplicaciones una mejor adaptación a los constantes cambios que les exige un escenario de colaboración.

Arquitectura desacoplada

Los sistemas CSCW/CSCL, tradicionalmente, han incrustado la lógica de los servicios de colaboración junto al de la lógica de las aplicaciones [45]. Este hecho ha traído consecuencias graves, como la disminución en el ciclo de vida de estos sistemas, debido a que complica seriamente su mantenimiento.

AORTA es una arquitectura que ofrece servicios de coordinación y conciencia de grupo de forma desacoplada. Este desacoplo es beneficioso porque permite separar oportunamente la lógica de colaboración de la lógica de las aplicaciones facilitando: la incorporación de servicios de coordinación y conciencia de grupo en aplicaciones existentes, el desarrollo de nuevas aplicaciones que incorporen dichos servicios y el mantenimiento tanto de aplicaciones como de servicios.

El desacoplo que AORTA ofrece está sustentado en el uso de una modificación del patrón *Command*. Esta modificación en conjunto con técnicas de Programación Orientada a Aspectos (AOP) es lo que permite a AORTA introducir, de forma desacoplada, servicios de coordinación y conciencia de grupo en las aplicaciones.

En el caso específico de AORTA, el uso de patrones de diseño para mediar el acoplamiento entre los servicios de colaboración y las aplicaciones ha tenido algunas implicaciones (los programadores deben tener ciertas consideraciones de diseño para el uso de AORTA).

A pesar de esto una de las aportaciones más importantes de este trabajo, sin duda, ha sido resaltar la importancia que tiene el uso de patrones de diseño en el ciclo de desarrollo de las aplicaciones CSCW/CSCL. Esta aseveración queda de manifiesto en dos ideas fundamentales:

- Aplicaciones basadas en patrones de diseño pueden ser más fácilmente integradas a marcos.
- Instruir a los programadores en el uso de patrones contribuye a reducir el ciclo de desarrollo de aplicaciones CSCW/CSCL.

Arquitectura orientada a acciones

La mayor parte de los sistemas CSCW/CSCL que ofrecen servicios de colaboración están basados en eventos de bajo nivel de abstracción (por ejemplo, eventos de interfaz). Desafortunadamente, en muchas ocasiones, estas abstracción representan un *déficit* de información porque no permiten, o muy difícilmente lo hacen, obtener la información relacionada con la acción que originó el evento (por ejemplo la operación que se quiere realizar, la autoría del evento, el objeto sobre el que se realiza la operación, el lugar de la operación, el tiempo de la operación, etc.)

AORTA, sin embargo, está basada en una aproximación que utiliza la acción como la mínima unidad lógica para el intercambio de información. Las acciones son abstracciones más cercanas al dominio de la colaboración que otras aproximaciones de más bajo de nivel como los eventos de interfaz.

Proveer un modelo orientado acciones es una de las contribuciones más destacadas del presente trabajo debido a que evidencia la necesidad de integrar modelos de información con niveles de abstracción más altos en marcos y aplicaciones CSCW/CSCL. Concretamente, en el caso de AORTA, el uso de acciones ha tenido aportaciones durante el ciclo de desarrollo de nuevas aplicaciones CSCW/CSCL disminuyendo la curva de aprendizaje de los programadores.

Utilizar acciones, desde luego, tiene algunas implicaciones en el desarrollo de las aplicaciones CSCW/CSCL. El uso de AORTA, por ejemplo, ha mostrado que los programadores deben hacer consideraciones de diseño para adecuar su código al manejo de acciones. Esta adecuación, que depende de la habilidad de cada programador, resulta fundamental para el correcto funcionamiento de una aplicación CSCW/CSCL.

6.2. Trabajo futuro

Desde luego, este trabajo ha dejado muchas líneas de investigación abiertas que denotan lo mucho que aún falta por hacer. Con respecto al trabajo realizado con AORTA, algunas de las tareas más interesantes por hacer a corto plazo son:

- **Más validación.** En el caso específico de AORTA contar con más casos prácticos sería ideal. Hasta ahora, AORTA sólo ha sido probada junto con un marco (ANTS) y una aplicación colaborativa (MagicPuzzle). Sin duda, uno de los logros futuros de este trabajo será la validación funcional de AORTA dentro de un contexto de marcos y aplicaciones más amplio.
- **Un lenguaje para la especificación de políticas.** Ofrecer un lenguaje para la creación de políticas, similar al que ofrecen otras propuestas como [19][44], mejoraría notablemente la flexibilidad de AORTA. Más aún, proveer este lenguaje junto con un grupo de herramientas que permitan a educadores y aprendices crear, editar y manejar de forma simple sus propias políticas sería una contribución notable para este trabajo.

Dentro de las nuevas líneas de investigación que se han abierto a raíz de su vinculación con este trabajo es posible destacar las siguientes:

- El uso de la Programación Orientada a Aspectos para facilitar la transición de aplicaciones monousuarios a aplicaciones colaborativas.
- La inclusión de acciones como los elementos primarios en el funcionamiento de otros marcos CSCW/CSCL.
- El desarrollo de nuevo middleware CSCW/CSCL. Específicamente, el middleware *peer to peer* (P2P) [62] puede ayudar a modelar y soportar escenarios de colaboración más flexibles y autónomos.

Aunado a estas ideas, otra línea de investigación de interés sería el estudio de una posible sistematización que facilite el tránsito del trabajo realizado en materia de computo colaborativo a escenarios completamente descentralizados. Por ejemplo, en el caso de AORTA, su arquitectura descentralizada y desacoplada facilitaría notablemente su transición a un escenario de estas características.

Bibliografía

[1] Baecker and Ronald M. *Readings in GroupWare and Computer-Supported Cooperative Work: Assisting Human-Human Collaboration.* Morgan Kaufmann Publishers Inc., 1994.

[2] Bartlett D. CORBA Component Model (CCM). 2001. http://www-106.ibm.com/developerworks/webservices/library/co-cjct6/index.html [visitada 2005/05].

[3] Beca L., Fox G., and Podgorny M. Component architecture for building web-based synchronous collaboration systems. In *Proceedings of the 8th Workshop on Enabling Technologies on Infrastructure for Collaborative Enterprises*, pages 108–113. IEEE Computer Society, 1999.

[4] Booch G. *Software components with Ada: structures, tools, and subsystems.* Benjamin Cummings, Menlo Park, California, 1987.

[5] Box D. *Essential COM.* Addison-Wesley, Redondo Beach, CA, 1997.

[6] Bronsard F., Bryan D., Kozaczynski W., Liongosari E., Ning J., and Wetterstrand J. Toward software plug-and-play. In *Proceedings of the 1997 symposium on Software reusability*, pages 19–29. ACM Press, 1997.

[7] Bruffee K. *Collaborative learning.* John Hopkings University Press, Baltimore, USA, 1993.

[8] Carzaniga A., Rosemblum D., and Wolf A. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, 2001.

[9] Chabert A., Grossman E., Jackson L., Pietrowiz S., and Seguin C. Java object-sharing in habanero. *Commun. ACM*, 41(6):69–76, 1998.

[10] Cohen D. *The Oxford Companion to the Mind*, chapter Behaviorism, page 71. Richard L. Gregory, 1987.

[11] Crawley R. A comparison of computer supported collective learning applications. Technical report, Dept. of Mechanical and Manufacturing Engineering, University of Brighton, 1997. http://www.brighton.ac.uk/cscl/jtap/paper2.htm [visitada 2005/05].

[12] Crnkovic I., Hnich B., Jonsson T., and Kiziltan Z. Specification, implementation, and deployment of components. *Commun. ACM*, 45(10):35–40, 2002.

[13] Crouch M. *ASP.NET and VB.NET Web Programming.* Addison-Wesley, Salt Lake City, Utah, 2002.

[14] Day M. What synchronous groupware needs: Notification services. Technical report, IBM Watson Research Center, 1997. In Proceedings of the 6th IEEE Workshop on Hot Topics in Operating Systems (HotOS-VI), Chatham, MA.

[15] Dodl N. Instructional groupware: Design considerations. In *In Proceedings of the 32nd Annual International Conference of the Association for the Development of Computer-Based Instructional Systems, Columbos, Ohio State University, USA*, pages 344–352. Dalton D.W., 1990.

[16] Dourish P. and Bellotti V. Awareness and coordination in shared workspaces. In *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, pages 107–114. ACM Press, 1992.

[17] Driscoll M. *Psychology of learning for instruction.* Prentice Hall, Boston, MA, 1994.

[18] D'Souza D. and Wills A. *Objects, Components, and Frameworks With Uml: The Catalysis Approach.* Addison-Wesley, 1998.

[19] Edwards W. Policies and roles in collaborative applications. In *Proceedings of the 1996 ACM conference on Computer supported cooperative work*, pages 11–20. ACM Press, 1996.

[20] Ellis C., Gibbs S., and Rein G. Groupware: some issues and experiences. *Commun. ACM*, 34(1):39–58, 1991.

[21] Ellis C. and Wainer J. A conceptual model of groupware. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 79–88. ACM Press, 1994.

[22] Elrad T., Filman R., and Bader A. Aspect-oriented programming: Introduction. *Commun. ACM*, 44(10):29–32, 2001.

[23] Galegher J. and Kraut R. Computer-mediated communication for intellectual teamwork: a field experiment in group writing. In *Proceedings of the 1990 ACM conference on Computer-supported cooperative work*, pages 65–78. ACM Press, 1990.

[24] Gamma E., Helm R., and Johnson R. *Design Patterns: Elements of Reusable Object Oriented Software.* Addison-Wesley, 1995.

[25] García P., Montala O., Pairot C., Rallo R., and Skarmeta A. Move: component groupware foundations for collaborative virtual environments. In *Proceedings of the 4th international conference on Collaborative virtual environments*, pages 55–62. ACM Press, 2002.

[26] García P., Skarmeta A., and Rallo R. Ants: A new collaborative learning framework. In *European Conference on Computer Supported Collaborative Learning*, 2001.

[27] Gosling J. and McGilton H. The Java Language Environment. 1996. http://java.sun.com/docs/white/langenv/index.html [visitada 2005/05].

[28] Groove Networks. Introduction to Groove, 2001. http://www.groovenetworks.com/pdf/introducinggroove.pdf [visitada 2001/3].

[29] Grundy J. and Hosking J. Engineering Plug-in Software Components to Support Collaborative Work. *Software Practice and Experience*, pages 983–1013, 2002.

[30] Gutwin C. and Greenberg S. Workspace awareness for groupware. In *Proceedings of Conference companion on human factors in computing systems*, pages 208–209. ACM Press, 1996.

[31] Gutwin C. and Greenberg S. The effects of workspace awareness support on the usability of real-time distributed groupware. *ACM Trans. Comput.-Hum. Interact.*, 6(3):243–281, 1999.

[32] Gutwin C., Stark G., and Greenberg S. Support for workspace awareness in educational groupware. In *The first international conference on Computer support for collaborative learning*, pages 147–156. Lawrence Erlbaum Associates, Inc., 1995.

[33] Hill J. and Gutwin C. Awareness support in a groupware widget toolkit. In *Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work*, pages 258–267. ACM Press, 2003.

[34] Holt and Anatol W. Diplans: a new language for the study and implementation of coordination. *ACM Trans. Inf. Syst.*, 6(2):109–125, 1988.

[35] Jacobson I., Christerson M., Jonsson P., and Overgaard P. *Object-Oriented Software Engineering: A Use Case Driven Approach.* 1992.

[36] Johnson P. and Johnson T. Consider the groupware: Design and group process impacts on communication in the electronic medium. In S. Hiltz and E. Kerr, editors, *Studies of Computer-Mediated Communications Systems: A Synthesis of the Findings*, number 16. Computerized Conferencing and

Communications Center, New Jersey Institute of Technology, Newark, New Jersey, EUA, 1981.

[37] Johnson R. Components, frameworks, patterns. In *Proceedings of the 1997 symposium on Software reusability*, pages 10–17. ACM Press, 1997.

[38] Johnson R. Frameworks = (components + patterns). *Commun. ACM*, 40(10):39–42, 1997.

[39] Jonassen D., Peck K., and Wilson B. *Learning with technology: a constructivist perspective*. Prentice Hall, 1999.

[40] Koschmann T. Computer support for collaboratlve learning: Design, theory, and research issues. *SIGCUE Outlook*, 21(3):1–2, 1992.

[41] Koschmann T. Toward a theory of computer support for collaborative learning. *Journal of the Learning Sciences*, 3(3):219–225, 1994.

[42] Krieger D. and Adler R. The emergence of distributed component platforms. *Computer*, 31(3):43–53, 1998.

[43] Lewandowski S. Frameworks for component-based client/server computing. *ACM Comput. Surv.*, 30(1):3–27, 1998.

[44] Li D. and Muntz R. Coca: collaborative objects coordination architecture. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work*, pages 179–188. ACM Press, 1998.

[45] Li D. and Muntz R. A collaboration specification language. In *Proceedings of the 2nd conference on Domain-specific languages*, pages 149–162. ACM Press, 1999.

[46] Lowy J. *Programming .Net components*. O'Relly, 2003.

[47] Malone T. and Crowston K. What is coordination theory and how can it help design cooperative work systems? In *Proceedings of the 1990 ACM conference on Computer-supported cooperative work*, pages 357–370. ACM Press, 1990.

[48] Malone T. and Crowston K. The interdisciplinary study of coordination. *ACM Comput. Surv.*, 26(1):87–119, 1994.

[49] Mantara Software. Elvin: Event notification wide-area system. http://elvin.dstc.edu.au/doc/index.html [visitada 2005/05].

[50] Mantei M., Baecker R., Sellen A., Buxton W., Milligan T., and Wellman B. Experiences in the use of a media space. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 203–208. ACM Press, 1991.

[51] Martinez A. *Método y modelo para el apoyo computacional a la evaluación en CSCL.* PhD thesis, Universidad de Valladolid, 2003.

[52] Martinez A., Dimitriadis Y., and De la Fuete P. *Computers and education. Towards a lifelong learning society*, chapter Contributions to analysis of interactions for formative evaluation in CSCL, pages 227–238. 2003.

[53] Martinez A., Dimitriadis Y., Rubia B., Gomez E., and De la Fuente P. Combining qualitative evaluation and social network analysis for the study of classroom social interactions. *Comput. Educ.*, 41(4):353–368, 2003.

[54] McDaniel S. and Brinck T. Awareness in collaborative systems: a chi 97 workshop. *SIGCHI Bull.*, 29(4):68–71, 1997.

[55] Microsoft. Component Object Model (COM), 2000. http://www.microsoft.com/com/resources/comdocs.asp.

[56] Microsoft. Microsoft.Net. 2003. http://www.microsoft.com/net/.

[57] Mühlenbrock M. *Action Based Collaboration Analysis for Group Learning.* Amsterdam, The Netherlands, 2001.

[58] Nierstrasz O. and Dami L. Component-oriented software technology. *Prentice Hall*, pages 3–28, 1995. Hertfordshire, UK.

[59] Object Management Group. Common Object Request Broker Architecture 3.0, 2002. http://www.omg.org/technology/documents/formal/corba_iiop.htm.

[60] Orfali R. and Harkey D. *Client Server Programming with Java and CORBA.* John Wiley & Sons, 1998.

[61] Orozco P., Asensio J., García P., Pairot C., and Dimitriadis Y. A Decoupled Architecture for Action-Oriented Coordination and Awareness Management in CSCL/W Frameworks. In *Groupware: Design, Implementation and Use: 10th International Workshop CRIWG 2004*, San Carlos, Costa Rica, 2004. Springer-Verlag.

[62] Pairot C., García P., and Gómez A. Dermi: A new distributed hash table-based middleware framework. *IEEE Internet Computing*, 8(3):74–84, 2004.

[63] Palo Alto Research Center. AOP: Aspect-Oriented Programming. http://www2.parc.com/csl/projects/aop/.

[64] Palo Alto Research Center. AspectJ. http://aspectj.org.

[65] Patterson J., Hill R., Rohall S., and Meeks S. Rendezvous: an architecture for synchronous multi-user applications. In *Proceedings of the 1990 ACM conference on Computer-supported cooperative work*, pages 317–328. ACM Press, 1990.

[66] Perret C. *La construcción de la inteligencia en la interacción social.* Visor Libros, 1984.

[67] Pinto M., Fuentes L., Fayad M., and Troya J. Separation of coordination in a dynamic aspect oriented framework. In *Proceedings of the 1st international conference on Aspect-oriented software development*, pages 134–140. ACM Press, 2002.

[68] Pozo I. *Aprendices y maestros*, chapter Las teorías del aprendizaje: De la asociación a la construcción, pages 60–64. Alianza Editorial, 2000.

[69] Pritchard J. *COM and CORBA side by side.* Addison-Wesley, 1999.

[70] Real Academia Española. Diccionario de la lengua española, 2001.

[71] Rogerson D. *Inside COM.* Microsoft Press, 1997.

[72] Roman E., Ambler S., and Jewell T. *Mastering Enterprise JavaBeans.* Robert Ipsen, 2002.

[73] Romero M., Osuna C., and Sheremetov L. Study and analysis of the working space conscience in c debate: a groupware application for collaborative debates. In *Proceedings of the Latin American conference on Human-computer interaction*, pages 107–115. ACM Press, 2003.

[74] Roschelle J. and Behrend S. The construction of shared knowledge in collaborative problem solving. In *Computer Supported Collaborative Learning*, pages 69–97. O Malley C. Springer-Verlag, Berlin, 1995.

[75] Roschelle J., Kaput J., and Stroup W. Scalable integration of educational software: Exploring the promise of component architectures. *Journal of Interactive Media in Education*, 1998.

[76] Roseman M. and Greenberg S. Groupkit: a groupware toolkit for building real-time conferencing applications. In *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, pages 43–50. ACM Press, 1992.

[77] Roseman M. and Greenberg S. Building real-time groupware with groupkit, a groupware toolkit. *ACM Trans. Comput.-Hum. Interact.*, 3(1):66–106, 1996.

[78] Sametinger J. *Software Engineering With Reusable Components.* Springer Verlag, 1997.

[79] Scardamalia M. and Bereiter C. *An Architecture for Collaborative Knowledge Building.*, chapter Computer-Based Learning Environments and Problem Solving. Heidelberg, FRG: Springer-Verlag, 1992.

[80] Schmidt K. and Simone C. Coordination mechanisms: Towards a conceptual foundation of CSCW systems design. *Computer Supported Cooperative Work*, 5(2/3):155–200, 1996.

[81] Slavin R. Cooperative learning and student achievement. Technical Report EORIG- 86-006, Office of Educational Research, Washington DC, USA, 1987.

[82] Sohlenkamp M. and Chwelos G. Integrating communication, cooperation, and awareness: the diva virtual office environment. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 331–343. ACM Press, 1994.

[83] Sun Microsystems. Java remote method invocation 1.4. http://java.sun.com/j2se/1.4.2/docs/guide/rmi/index.html.

[84] Sun Microsystems. Enterprise Java Beans Specification 2.1, 2003. http://java.sun.com/products/ejb/docs.html.

[85] Sun Microsystems. Java 2 Platform Enterprise Edition, 2003. http://java.sun.com/j2ee/1.4/download.html.

[86] Sun Microsystems. Java Shared Data Toolkit, 2003. http://java.sun.com/products/java-media/jsdt/index.jsp.

[87] Sun Microsystems. JavaBeans Specification 1.0, 2003. http://java.sun.com/products/javabeans/docs/spec.html.

[88] Suthers D. Architectures for Computer Supported Collaborative Learning. In *Proceedings of the IEEE International Conference on Advanced Learning Technologies*, Wisconsin, USA, 2001.

[89] Szyperski C. *Component Software - Beyond Object Oriented Programming.* Adddison Wesley, 1998.

[90] Szyperski C. Components and objects together, 1999. Software Development Magazine - http://www.sdmagazine.com/columnists/szyperski/.

[91] Szyperski C. Component technology: what, where, and how? In *Proceedings of the 25th International Conference on Software Engineering*, pages 684–693. IEEE Computer Society, 2003.

[92] Szyperski C. and Gough J. Special issues in object oriented programming. In *Workshop on Component Oriented Programming*, pages 99–114. Alpha Books, Bristol, 1996.

[93] Waldegg G. El uso de las nuevas tecnologías para la enseñanza y el aprendizaje de las ciencias. In *Revista Electrónica de Investigación Educativa*, volume 4, 2002.

[94] Yang G. A uniform meta-model for modeling integrated cooperation. In *Proceedings of the 2002 ACM symposium on Applied computing*, pages 322–328. ACM Press, 2002.

Apéndice

Interfaz de Programación de AORTA

Package

aorta.application

aorta.application
Class Action

```
java.lang.Object
   |
   +-aorta.application.Action
```

All Implemented Interfaces:
java.io.Serializable

---

public class **Action**
extends java.lang.Object
implements java.io.Serializable

Action class represents the abstract super class for any action. All action classes must extend this class.

Constructors

Action

`public Action()`

Constructs a new Action object.

Methods

getDate

`public java.util.Date getDate()`

Returns the date of this action.

Returns:
The date of this action.

---

getStringDate

`public java.lang.String getStringDate()`

Returns the string representation of the date of this action.

Returns:
The date object of this action (string format).

---

setUser

`public void setUser(java.lang.String user)`

Sets the user of this action.

Parameters:
`user` - The user to be set.

(continued on next page)

(continued from last page)

getUser

`public java.lang.String getUser()`

Returns the user of this action. The user who has performed the action

Returns:
The user of this action.

setAllowed

`public void setAllowed(boolean result)`

Sets the execution result of an action.

Parameters:
`result` - The result to be assigned to this action.

isAllowed

`public boolean isAllowed()`

Returns the execution result of this action.

Returns:
`true` if an action was successfully executed or `false` if the action execution was not allowed.

getObject

`public java.lang.String getObject()`

Returns the object of this action. The object on which the action is performed.

Returns:
The object on which the action is performed.

getName

`public java.lang.String getName()`

Returns the name of this action.

Returns:
The string that represents the name of this action.

setDate

`public void setDate(java.util.Date date)`

Sets the date when this action was performed.

Parameters:
`date` - The date when this action was performed.

setName

`public void setName(java.lang.String name)`

Sets the name of this action.

Parameters:
`name` - The name of this action.

setObject

`public void **setObject**(java.lang.String object)`

Sets the object of this action. The object on which the action is performed.

Parameters:

`object` - The object on which the action is performed.

setMessage

`public void **setMessage**(java.lang.String message)`

getMessage

`public java.lang.String **getMessage**()`

Sets the object of this action. The object on which the action is performed.

Parameters:

`object` - The object on which the action is performed.

aorta.application
Class ActionExecutionEngine

```
java.lang.Object
   |
   +-aorta.application.ActionExecutionEngine
```

public class **ActionExecutionEngine**
extends java.lang.Object

The Action Execution Engine is the class responsable of executing actions. An application requests the execution to the Action Execution Engine.

Constructors

ActionExecutionEngine

```
public ActionExecutionEngine()
```

Constructs a new ActionExecutionEngine object

Methods

execute

```
public void execute(IAction action)
  throws IllegalActionException
```

Executes an action.

Parameters:

`action` - The action to be executed.

aorta.application
Interface IAction

public interface **IAction**

Interface for Actions that must be executed through the Action Executed Engine.

Fields

name

`public static final java.lang.String` **`name`**

object

`public static final aorta.application.IObject` **`object`**

user

`public static final java.lang.String` **`user`**

Methods

getObject

`public IObject` **`getObject`**`()`

Returns the object of this action. The object on which the action is performed.

Returns:

The object on which the action is performed.

setObject

`public void` **`setObject`**`(IObject object)`

Sets the object of this action. The object on which the action is performed.

Parameters:

`object` - The object on which the action is performed.

getName

`public java.lang.String` **`getName`**`()`

Returns the name of this action.

Returns:

The string that represents the name of this action.

setName

`public void **setName**(java.lang.String name)`

Sets the name of this action.

Parameters:

`name` - The name of this action.

getUser

`public java.lang.String **getUser**()`

Returns the user of this action. The user who has performed the action

Returns:

The user of this action.

setUser

`public void **setUser**(java.lang.String user)`

Sets the user of this action.

Parameters:

`user` - The user to be set.

execute

`public void **execute**()`

Executes the logic of this action.

aorta.application
Interface IObject

public interface **IObject**

Interface for Objects that must be managed through AORTA.

Fields

id

```
public static final java.lang.String id
```

name

```
public static final java.lang.String name
```

Methods

setId

```
public void setId(java.lang.String id)
```

Sets the string identifier of this object.

Parameters:

`id` - The string identifier of this object.

getId

```
public java.lang.String getId()
```

Returns the string identifier of this object.

Returns:

The string that represents the identifier of this object.

getName

```
public java.lang.String getName()
```

Returns the name of this object.

Returns:

The name of this object.

setName

```
public void setName(java.lang.String name)
```

Sets the name of this object.

(continued from last page)

Parameters:

`name` - The name of this object.

aorta.application
Interface IUser

public interface **IUser**

Interface for Users that perform actions through AORTA.

Fields

id

```
public static final java.lang.String id
```

name

```
public static final java.lang.String name
```

Methods

setId

```
public void setId(java.lang.String id)
```

Sets the string identifier of this user.

Parameters:

`id` - The string identifier of this user.

getId

```
public java.lang.String getId()
```

Returns the string identifier of this user.

Returns:

The string that represents the identifier of this usert.

getName

```
public java.lang.String getName()
```

Returns the name of this user.

Returns:

The name of this user.

setName

```
public void setName(java.lang.String name)
```

Sets the name of this user.

(continued from last page)

Parameters:

`name` - The name of this user.

Package

aorta.collaboration.awareness

aorta.collaboration.awareness
Class AMView

```
java.lang.Object
   |
   +-java.awt.Component
         |
         +-java.awt.Container
               |
               +-java.awt.Window
                     |
                     +-java.awt.Frame
                           |
                           +-javax.swing.JFrame
                                 |
                                 +-aorta.collaboration.awareness.AMView
```

public class **AMView**
extends javax.swing.JFrame

AMView (Awareness Manager View) class is an administration tool to change awareness policies at run time.

Constructors

AMView

`public AMView(Place oPlace)`

Constructs a new AMView object.

Methods

main

`public static void main(java.lang.String[] args)`

Main method of AMview class. This method allows to run the class with the Java interpreter.

aorta.collaboration.awareness
Class AwarenessManager

```
java.lang.Object
   |
   +-aorta.collaboration.awareness.AwarenessManager
```

All Implemented Interfaces:
ActionNotificationListener

---

public class **AwarenessManager**
extends java.lang.Object
implements ActionNotificationListener

AwarenessManager class represents one of the main elements of AORTA framework. This class performs all the awareness related tasks.

Constructors

AwarenessManager

`public` **`AwarenessManager`**`()`

Constructs a new AwarenessManager object.

Methods

notifyAction

`public void` **`notifyAction`**`(Action action)`

Notifies the performing of an action to the awareness manager. The awareness manager then evaluates if the action must be notified to other participants.

Parameters:
`action` - The action performed.

---

setPolicy

`public void` **`setPolicy`**`(java.lang.String policy)`

Sets an awareness policy. This is the policy that the awareness manager uses to evaluate actions.

Parameters:
`policy` - The policy to be set.

---

destroyPolicy

`public void` **`destroyPolicy`**`()`

Destroys the current awareness policy.

---

initPolicy

`public void` **`initPolicy`**`()`

(continued from last page)

Performs the initialization of a new awareness policy.

loadPolicy

`public void loadPolicy()`

Loads the set awareness policy.

getAdministrator

`public java.lang.String getAdministrator()`

Returns the awareness manager user. This is the only user allowed to perform any awareness change.

Returns:

The user allowed to perform awareness changes.

getPolicies

`public java.lang.String getPolicies()`

Returns a list with all available awareness policies.

Returns:

The list with all available awareness policies.

eventArrived

`public void eventArrived(java.util.Hashtable event)`

Recives action notifications if the awareness manager is subscribed to it.

Parameters:

`event` - The event wich contains the notified action.

aorta.collaboration.awareness
Class IllegalNotificationException

```
java.lang.Object
   |
   +-java.lang.Throwable
        |
        +-java.lang.Exception
             |
             +-aorta.collaboration.awareness.IllegalNotificationException
```

public class **IllegalNotificationException**
extends java.lang.Exception

This exception is thrown when an attempt is made to notify any user who is not allowed to receive notifications.

Constructors

IllegalNotificationException

`public IllegalNotificationException()`

Constructs a new instance of IllegalNotificationException.

Package

aorta.collaboration.awareness.repository

aorta.collaboration.awareness.repository
Class Everybody

```
java.lang.Object
   |
   +-aorta.collaboration.awareness.repository.Everybody
```

All Implemented Interfaces:
IPolicy

---

public class **Everybody**
extends java.lang.Object
implements IPolicy

Everybody is a policy class which allows to notify all participants about any action.

Constructors

Everybody

`public Everybody()`

Methods

init

`public void init(Place conn)`

Performs the initialization of the Everybody class. This method is call each time a Everybody policy is created.

Parameters:
`conn` - The connection to the Ants framework.

---

destroy

`public void destroy()`

Destroys the current awareness policy. This method contains resources that must be process before the policy destruction.

---

checkNotification

`public boolean checkNotification(Action action)`

Evaluates if an action should be notified or not to other participants.

Parameters:
`action` - The action to be evaluated.

Returns:
`true` if the notification is allowed and `false` if not.

aorta.collaboration.awareness.repository
Interface IPolicy

All Known Implementing Classes:
Everybody, Leader

public interface **IPolicy**

A developer who wishes to create a new awareness policy must implement this interface

Methods

checkNotification

```
public boolean checkNotification(Action action)
  throws IllegalNotificationException
```

Evaluates if an action should be notified or not to other participants.

Parameters:
`action` - The action to be evaluated.

Returns:
`true` if the notification is allowed and `false` if not.

Throws:
`This` - exception is thrown when an attempt is made to notify a user which is not allowed to receive notifications.

destroy

```
public void destroy()
```

Destroys the current awareness policy. This method contains resources that must be process before the policy destruction.

init

```
public void init(Place conn)
```

Performs the initialization of a new awareness policy. This method is call each time a policy is created.

Parameters:
`conn` - The oPlace object is an Ants framework connection object.

aorta.collaboration.awareness.repository
Class Leader

```
java.lang.Object
   |
   +-aorta.collaboration.awareness.repository.Leader
```

All Implemented Interfaces:
IPolicy

---

public class **Leader**
extends java.lang.Object
implements IPolicy

Leader is a policy class which only allows to notify actions to leader participants (participants with a leader role).

Constructors

Leader

`public Leader()`

Methods

init

`public void init(Place conn)`

Performs the initialization of the Leader class. This method is call each time a Leader policy is created.

Parameters:
`conn` - The connection to the Ants framework.

---

destroy

`public void destroy()`

Destroys the current awareness policy. This method contains resources that must be process before the policy destruction.

---

checkNotification

`public boolean checkNotification(Action action)`

Evaluates if an action should be notified or not to other participants.

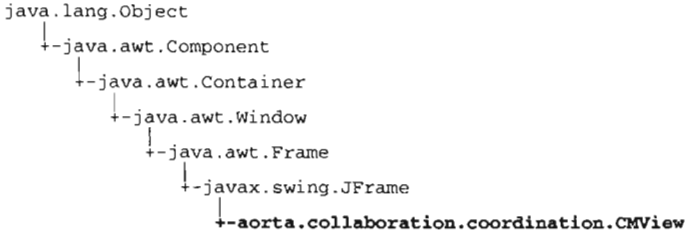
Parameters:
`action` - The action to be evaluated.

Returns:
`true` if the notification is allowed and `false` if not.

Package

aorta.collaboration.coordination

aorta.collaboration.coordination
Class CMView



public class **CMView**
extends javax.swing.JFrame

CMView (Coordination Manager View) class is an administration tool to change coordination policies at run time.

Constructors

CMView

`public CMView(Place oPlace)`

Constructs a new CMView object.

Methods

main

`public static void main(java.lang.String[] args)`

Main method of CMview class. This method allows to run the class with the Java interpreter.

init

`public void init()`

This method is invoked each time that a new CMView object is created.

aorta.collaboration.coordination
Class CoordinationManager

```
java.lang.Object
   |
   +--aorta.collaboration.coordination.CoordinationManager
```

public class **CoordinationManager**
extends java.lang.Object

CoordinationManager class represents one of the main elements of AORTA framework. This class performs all the coordination related tasks.

Constructors

CoordinationManager

`public CoordinationManager()`

Constructs a new CoordinationManager object.

Methods

checkAction

`public void checkAction(Action action) throws IllegalActionException`

Evaluates if an action should be performed or not

Parameters:
`action` - The action to be evaluated.

loadPolicy

`public void loadPolicy()`

Loads the set coordination policy.

setPolicy

`public void setPolicy(java.lang.String policy)`

Sets a coordination policy. This is the policy that the CoordinationManager uses to evaluate actions.

Parameters:
`policy` - The policy to be set.

getAdministrator

`public java.lang.String getAdministrator()`

Returns the coordination manager user. This is the only user allowed to perform any coordination change.

Returns:
The user allowed to perform coordination changes.

getPolicies

`public java.lang.String` **`getPolicies`**`()`

Returns a list with all available coordination policies.

Returns:

The list with all available coordination policies.

destroyPolicy

`public void` **`destroyPolicy`**`()`

Destroys the current coordination policy.

initPolicy

`public void` **`initPolicy`**`()`

Performs the initialization of a new coordination policy.

aorta.collaboration.coordination
Class IllegalActionException

```
java.lang.Object
   |
   +-java.lang.Throwable
      |
      +-java.lang.Exception
         |
         +-aorta.collaboration.coordination.IllegalActionException
```

public class **IllegalActionException**
extends java.lang.Exception

This exception is thrown when an attempt is made to perform an illegal action.

Constructors

IllegalActionException

`public IllegalActionException()`

Constructs a new instance of IllegalActionException.

Package

aorta.collaboration.coordination.repository

aorta.collaboration.coordination.repository
Interface IPolicy

public interface **IPolicy**

A developer who wishes to create a new coordination policy must implement this interface

Methods

checkAction

```
public boolean checkAction(Action action)
  throws IllegalActionException
```

Evaluates if the execution of an action is legal or not.

Parameters:
`action` - The action to be evaluated.

Returns:
`true` if the action is allowed and `false` if not.

Throws:
`This` - exception is thrown when an attempt is made to perform an action wich is not allowed.

destroy

```
public void destroy()
```

Destroys the current coordination policy. This method contains resources that must be process before the policy destruction.

init

```
public void init(Place conn,
        CoordinationManager oCM)
```

Performs the initialization of a new coordination policy. This method is call each time a policy is created.

Parameters:
`conn` - A connection for Ants framework

aorta.collaboration.coordination.repository
Class Leader

```
java.lang.Object
   |
   +-aorta.collaboration.coordination.repository.Policy
      |
      +-aorta.collaboration.coordination.repository.Leader
```

public class **Leader**
extends Policy

Leader is a coordination policy class which allows one user (the leader) to decide who is allowed to perform the next action.

Constructors

Leader

```
public Leader()
```

Methods

init

```
public void init(Place conn,
          CoordinationManager oCM)
```

Performs the initialization of the Leader policy class. This method is call each time a Leader policy is created.

Parameters:
`conn` - The connection to the Ants framework.

destroy

```
public void destroy()
```

Destroys this policy. This method contains resources that must be processed before the policy destruction.

checkAction

```
public boolean checkAction(Action action)
```

Evaluates if the execution of an action is legal or not (a leader policy rules).

Parameters:
`action` - The action to be evaluated.

Returns:
`true` if the action is allowed and `false` if not.

Throws:
`This` - exception is thrown when an attempt is made to perform an action wich is not allowed.

(continued from last page)

eventArrived

`public void` **`eventArrived`**`(java.util.Hashtable event)`

Recives coordination state notifications.

Parameters:

`event` - The event with the new coordination state.

aorta.collaboration.coordination.repository
Class Token

```
java.lang.Object
   |
   +-aorta.collaboration.coordination.repository.Policy
      |
      +-aorta.collaboration.coordination.repository.Token
```

public class **Token**
extends Policy

Token is a coordination policy class wich allows only one user to perform an action on the same object. Differents users can only perform their actions on differents objects. If a user performs an illegal action more than five times he/she will not be able to perform more actions.

Constructors

Token

```
public Token()
```

Methods

init

```
public void init(Place conn,
          CoordinationManager oCM)
```

Performs the initialization of the Token policy class. This method is call each time a Token policy is created.

Parameters:
`conn` - The connection to the Ants framework.

destroy

```
public void destroy()
```

Destroys this policy. This method contains resources that must be processed before the policy destruction.

checkAction

```
public boolean checkAction(Action action)
```

Evaluates if the execution of an action is legal or not (a token policy rules).

Parameters:
`action` - The action to be evaluated.

Returns:
`true` if the action is allowed and `false` if not.

Throws:
`This` - exception is thrown when an attempt is made to perform an action wich is not allowed.

eventArrived

`public void` **`eventArrived`**`(java.util.Hashtable event)`

Recives coordination state notifications.

Parameters:

`event` - A event with the coordination state notification.

aorta.collaboration.coordination.repository
Class Turn

```
java.lang.Object
   |
   +--aorta.collaboration.coordination.repository.Policy
         |
         +--aorta.collaboration.coordination.repository.Turn
```

public class **Turn**
extends Policy

Turn is a coordination policy class wich allows to perform a second action only when the other participants have performed an action too.

Constructors

Turn

```
public Turn()
```

Methods

init

```
public void init(Place conn,
          CoordinationManager oCM)
```

Performs the initialization of the Turn policy class. This method is call each time a Turn policy is created.

Parameters:
`conn` - The connection to the Ants framework.

destroy

```
public void destroy()
```

Destroys this policy. This method contains resources that must be processed before the policy destruction.

checkAction

```
public boolean checkAction(Action action)
```

Evaluates if the execution of an action is legal or not (a turn policy rules).

Parameters:
`action` - The action to be evaluated.

Returns:
`true` if the action is allowed and `false` if not.

Throws:
`This` - exception is thrown when an attempt is made to perform an action wich is not allowed.

eventArrived

```
public void eventArrived(java.util.Hashtable event)
```

Recives coordination state notifications.

Parameters:

`event` - A event with the coordination state notification.

Package

aorta.communication

aorta.communication
Interface ActionNotificationListener

All Known Implementing Classes:
AwarenessManager

---

public interface **ActionNotificationListener**

This interface must be implemented by any application to receive awaress notifications.

---

aorta.communication
Class ActionNotificationService

```
java.lang.Object
   |
   +-aorta.communication.ActionNotificationService
```

public class **ActionNotificationService**
extends java.lang.Object

ActionNotificationService class allows to notify actions (through the network) to other participants.

Constructors

ActionNotificationService

`public ActionNotificationService()`

Methods

subscribeAwarenessNotification

`public void subscribeAwarenessNotification(ActionNotificationListener anl, java.lang.String session)`

Creates a subscription to the awareness notification service of AORTA

Parameters:

`anl` - An AwarenessNotificationListener object to be subscribed.

unSubscribeAwarenessNotification

`public void unSubscribeAwarenessNotification()`

Destroys the subscription to the awareness notification service of AORTA

subscribeCoordinationStateNotification

`public void subscribeCoordinationStateNotification(ActionNotificationListener anl, java.lang.String session)`

Creates a subscription to the coordination notification service of AORTA

Parameters:

`rel` - A RemoteEventListener object to be subscribed.

unSubscribeCoordinationStateNotification

`public void unSubscribeCoordinationStateNotification()`

Destroys the subscription to the coordination notification service of AORTA

(continued from last page)

notifyAwarenessAction

```
public void notifyAwarenessAction(Action action)
```

Notifies an awareness action (through the network) to other participants.

Parameters:

`action` - A action to be notified.

notifyCoordinationState

```
public void notifyCoordinationState(java.lang.String user,
          java.lang.String subject,
          java.lang.String content)
```

Notifies a new coordination state (through the network) to other participants.

Parameters:

`action` - A action to be sent.
`subject` - A subject for the notification.
`content` - A content for the notification.

aorta.communication
Class Message

```
java.lang.Object
   |
   +-aorta.communication.Message
```

All Implemented Interfaces:
java.io.Serializable

---

public class **Message**
extends java.lang.Object
implements java.io.Serializable

Message class contains the information needed to notify actions (through the network)

Constructors

Message

```
public Message()
```

Methods

setFrom

```
public void setFrom(java.lang.String from)
```

Sets the user who sends the message.

Parameters:
`from` - The user who sends the message.

---

getAttachment

```
public java.lang.Object getAttachment()
```

Returns the attachment object of this message.

Returns:
The attachment object of this message.

---

getSubject

```
public java.lang.String getSubject()
```

Returns the subject of this message.

Returns:
The subject of this message.

(continued from last page)

setAttachment

`public void setAttachment(java.lang.Object attach)`

Sets an attachment object.

Parameters:

`attach` - A attachment for the message.

getFrom

`public java.lang.String getFrom()`

Returns the user who sends this message.

Returns:

The user who sends this message.

setSubject

`public void setSubject(java.lang.String subject)`

Sets the message subject.

Parameters:

`subject` - A subject for the message.

getContent

`public java.lang.String getContent()`

Returns the content of this message.

Returns:

The content of this message.

setContent

`public void setContent(java.lang.String content)`

Sets the message content.

Parameters:

`content` - The content to be set.